ADAM-4500 Series

Ethernet-enabled Communication
Controller with I/O Modules

User's Manual

Copyright Notice

This document is copyrighted, 1997, by Advantech Co., Ltd. All rights are reserved. Advantech Co., Ltd., reserves the right to make improvements to the products described in this manual at any time without notice.
No part of this manual may be reproduced, copied, translated or transmitted in any form or by any means without the prior written permission of Advantech Co., Ltd. Information provided in this manual is intended to be accurate and reliable. However, Advantech Co., Ltd. assumes no responsibility for its use, nor for any infringements upon the rights of third parties, which may result from its use.

Acknowledgments

ADAM is a trademark of Advantech Co., Ltd.
IBM and PC are trademarks of International Business Machines Corporation.

Edition 2.0
May 2008

Table of Contents

Chapter 1 System Overview……..…….……. 1-1

1.1 Introduction .....…….....……..…… 1-2
1.2 Features .....…….....…….....……..…….. 1-3
1.3 ADAM-4501/4501D Controllers Specification……..1-7

Chapter 2 Installation Guidelines.....…….....……..... 2-1

2.1 System Requirements……..…..... 2-2
2.2 Hardware Installation ……..…..... 2-3
2.3 System Wiring and Connections.....…….....…….... 2-5
2.4 Software Installation…..…….....……..... 2-12

Chapter 3 I/O Modules .....…….....……..... 3-1

3.1 System Hardware Configuration……..…..... 3-2
3.2 Install Utility on Host PC……..…..... 3-2
3.3 ADAM-4500 Series Utility Overview..……..... 3-6
3.4 Initialize Drive D to Default Settings……..…..... 3-11
3.5 Configure IP Address and HTTP/FTP User/Password…….. 3-15
3.6 Download and Run Application Program Automatically
After Boot Up ……..... 3-18
3.7 Backup Drive D as Image File ……..... 3-23
3.8 Restore Drive D from Image File ….....…….... 3-27

Chapter 4 Guidelines for Network Functions.....…….....……..... 4-1

4.1 FTP Server.....…….....……..... 4-6
4.2 HTTP Server.....…….....……..... 4-9
4.3 Send Mail ……..…..... 4-24
4.4 Modbus/TCP Server ……..…..... 4-31
4.5 Modbus/TCP Client .....……..…..... 4-38
4.6 Modbus/RTU Slave ……..…..... 4-42
4.7 Modbus/RTU Master ……..…..... 4-49
4.8 TCP Server and Client ……..…..... 4-53

4.9 UDP Connection 4-66
4.10 FTP Client 4-75

Chapter 5 Programming and Function Library 5-1

5.1 Introduction 5-2
5.2 Category of Function Libraries 5-7
5.3 Function Library Description 5-12

Chapter 6 Sockets Utility 6-1

Chapter 7 HTTP and FTP Server Application..... 7-1

Appendix A COM Port Register Structure A-1

Appendix B RS-485 Network B-1

Appendix C Grounding Reference C-1

1

System Overview

1.1 Introduction

Standalone Data Acquisition and Control System

As the growth of PC-based technology, Advantech PC-based Programmable Controllers have been widely applied in variety of industrial automation applications. Shrunk from the original ADAM-5510 series controller, the ADAM-4500 Series Controller
is a new series of stand-alone programmable communication controller. It does not only support high memory capacity, user- friendly configuration tool, rich serial communication interfaces,
but also support Ethernet port available and original libraries on specific models. Applying the ADAM-4500 Series Controller, the C programmers would be able to handle any complex task easily.

The ADAM-4500 Series Controller is a compact-sized Ethernet- enabled communication controller under x-86 CPU architecture. It supports not only Ethernet interface but also 4 serial ports, which lets ADAM-4500 Series Controller be very suitable for industrial communication and control applications. The Ethernet-enabled features include built-in HTTP Server, FTP Server and E-mail functions. The modularized I/O design provides high flexibility for versatile application requirements. ADAM-4500 Series Controller also supports rich Modbus function libraries including Modbus/RTU (Master and Slave) and Modbus/TCP (Server and Client) function libraries.

The ADAM-4500 Series Controller includes following models:

- **ADAM-4501** Ethernet-Enabled Communication Controller with 8 DI/O
- **ADAM-4501D** Ethernet-Enabled Communication Controller with 8 DI/O and LED Display

1.2 Features

The system of ADAM-4500 Series Controller consists of two major components: the main unit and I/O modules. The main unit includes the communication ports, CPU ….and so on. The I/O Module of ADAM-4501/4501D includes the 8 digital I/O channels. Besides, the ADAM-4501D also includes 7-segment display to show needed information.

1.2.1 Control flexibility with C programming

The ADAM-4500 Series Controller includes an 80188 CPU and a built-in ROM-DOS operating system. It can be used in a way similar to how one uses an x86 PC in the office. Programmers in C can write and compile applications in Borland C 3.0 and download to the ADAM-4500 Series Controller. Given the prevalence of C language programming tools, this is a distinct advantage for many users and can result in a very short learning curve and very modest training expense requirements.

1.2.2 RS-232/485 communication ability

The ADAM-4500 Series Controller has four serial communication ports, giving it excellent communication abilities. This facilitates its ability to control networked devices. The communication ports of different models are listed as below table.

	ADAM-4501/4501D
COM1	RS-232(Full modem signal)
COM2	RS-485
COM3	RS-485
COM4/Prog	RS-485/RS-232

Table 1-1 Communication Ports of ADAM-4501/4501D Controller

ADAM-4501/4501D COM1 is a dedicated RS-232 port, COM2 and COM3 are dedicated RS-485 port, and the fourth communication port is shared by COM4 and Programming port. It is a selectable port by using jumper.

These four ports allowed the ADAM-4501/4501D to satisfy diverse communication and integration demands. Programming port is for downloading or transferring executable programs from a host PC to ADAM-4500 Series Controller. It can also be used as an RS-232 communication port (Refer to section 2.2.1 to see how to configure COM4 as Programming port or standard RS-232 communication port). Please refer to following figure to check the location of COM ports.

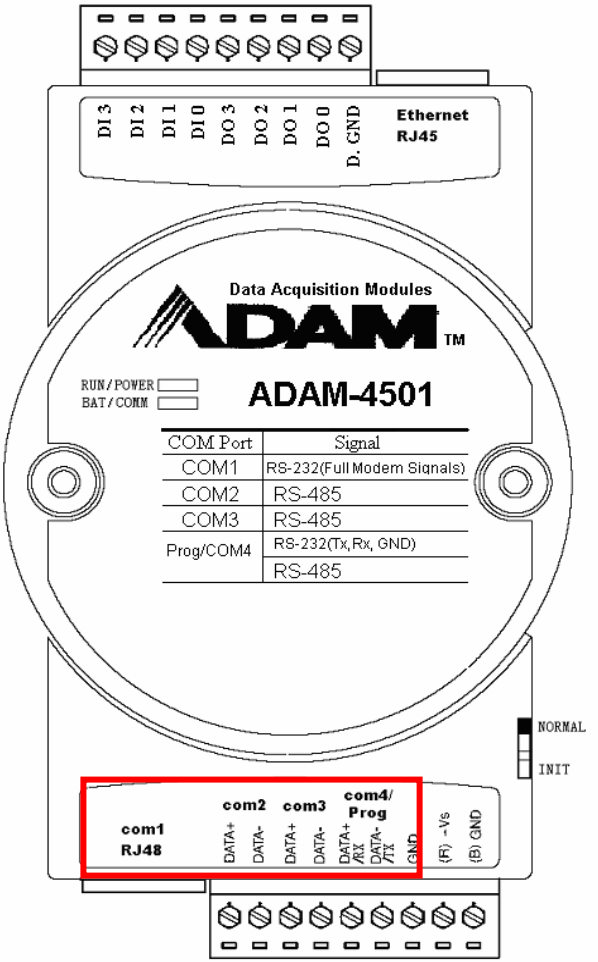


Figure 1-1 ADAM-4501 Communication Ports

1.2.3 Versatile Protocols of Communication Function Libraries

The communication protocol of the ADAM-4500 Series Controller is user-defined and there are library functions of MODBUS/RTU protocol and MODBUS/TCP protocol available for users. Of course, users can implement ASCII-based command and response protocol by themselves. The function libraries include following protocols.

- MODBUS/RTU Master Function for connecting to remote I/O modules via RS-485 port
- MODBUS/RTU Slave Function for connecting to HMI/SCADA software via RS-485 port
- MODBUS/TCP Server Function for connecting to HMI/SCADA software via Ethernet port.
- MODBUS/TCP Client Function for connecting to Ethernet- enabled remote I/O modules via Ethernet port.

1.2.4 Built-in ROM and RAM disk for programming

The ADAM-4500 Series Controller has built-in Flash Memory and SRAM for file downloading, system operation and data storage. It provides 1MB file system (960 KB free for users to download programs). There are also 640KB SRAM to provide the memory needed for efficient application operation and file transfer. Moreover, users are allowed to decide the battery backup memory size up to 384KB in the SRAM.

1.2.5 Built-in real-time clock and watchdog timer

The micro-controller also includes a real-time clock and watchdog timer. The real-time clock records events while they occur. The watchdog timer is designed to automatically reset the microprocessor if the system fails. This feature greatly reduces the level of maintenance required and makes the ADAM-4500
Series Controller ideal for use in applications which required a high level of system stability.

1.2.6 Built-in Ethernet Port

The Ethernet port on ADAM-4500 Series Controller can perform powerful function as following:

- FTP Server and Client Function
- Web Server Function
- Send Mail Function
- TCP and UDP Connection by Sockets

1.3 ADAM-4501/4501D Controllers Specification

1.3.1 System

- CPU: 16-bit microprocessor
- Memory:

  1.5MB Flash memory
  - 256KB system Disk (Drive C: Read Only)
  - 256KB flash memory (Accessed by Function LIB)
  - 1024KB file system, 960KB for user applications (Drive D: Read/Write)

  640KB SRAM
  - up to 384KB with battery backup (Accessed by Function LIB)
  -

- Operating System: ROM-DOS (MS-DOS 6.22 Compatible)
- Real-time Clock: yes
- Watchdog Timer: yes
- RS-232 interface: COM1
- RS-485 interface: COM2, COM3
- RS-232/485 interface: Programming Port & COM4 (Select by jumper setting)
- LAN port x 1: 10/100Base-T
- On-board I/O Capacity:

  Digital Input 4 Channels
  Dry Contact:
  Logic level 0: Close to GND
  Logic level 1: Open
  Wet Contact:
  Logic level 0: +2 V max.
  Logic level 1: 4 V ~ 30 V

  Digital Output 4 Channels
  Open Collector to +40 V, 200 mA (maximum load)

1.3.2 RS-232 interface (COM1)

- Signals: TxD, RxD, RTS, CTS, DTR, DSR, DCD, RI, GND
- Mode: Asynchronous full duplex, point to point
- Connector: DB-9 pin
- Transmission speed: Up to 115.2 Kbps
- Max transmission distance: 50 feet (15.2 m)

1.3.3 RS-485 interface (COM2 & COM3)

- Signals: DATA+, DATA-
- Mode: Half duplex, multi-drop
- Connector: Screw terminal
- Transmission speed: Up to 115.2 Kbps
- Max transmission distance: 4000 feet (1220 m)

1.3.4 RS-485/232 interface (COM4 and programming port)

- RS-232/485 Mode Selectable (Select by jumper setting)
- RS-232 Mode (**Programming port**): Full duplex, point to point
  Signals: TxD, RxD, GND
- RS-485 Mode: Half duplex, multi-drop
  Signals: DATA+, DATA-
- Connector: Screw terminal
- Transmission speed: Up to 115.2 Kbps
- Max transmission distance:
  RS-232: 50 feet (15.2 m)
  RS-485: 4000 feet (1220 m)

1.3.5 Power

- Unregulated +10 to +30 VDC
- Power consumption: 2.0 W (Typically)

1.3.6 Mechanical

- Case: ABS and PC
- Plug-in screw terminal block: two 9-pin plug in screw terminals
  Accepts #14~22 AWG (0.6~1.6 mm2)

1.3.7 Environment

- Operating temperature: -10° to 70° C (14° to 158° F)
- Storage temperature: -25° to 85° C (-13° to 185° F)
- Humidity: 5 to 95 %, non-condensing
- Atmosphere: No corrosive gases

Note: Equipment will operate below 30% humidity. However, static electricity problems occur much more frequently. Make sure you take adequate precautions when you touch the equipment.

1.3.8 Dimensions

The following diagrams show the dimensions of the system unit and an I/O unit. All dimensions are in millimeters.

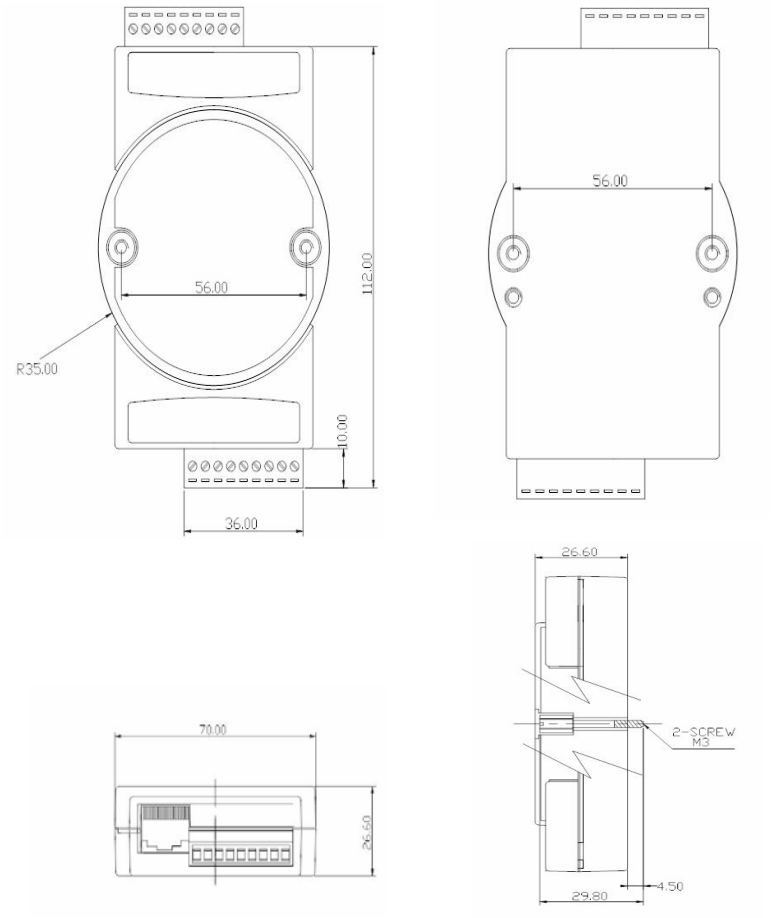


Figure 1-2 ADAM-4500 Series Controller Dimension

1.3.9 LED Status

There are four LED lights on the ADAM-4500 Series. The **PWR**, **RUN** and **COMM** LED can be controlled using ADAM-4500 Series library LED OFF and LED ON (refer to section 5.3.1). **BATT** LED is the battery status indicator. This LED will be on whenever the SRAM backup battery is low.

1.3.10 ADAM-4500 Series Controller System Architecture

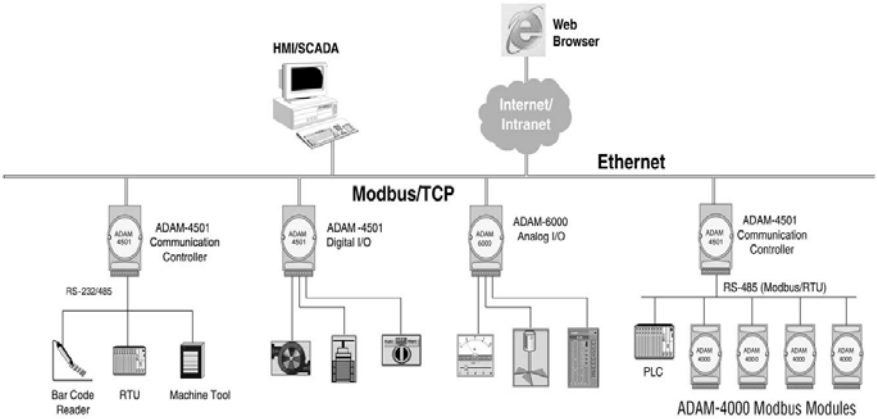


Figure 1-3 ADAM-4500 Series Controller System Architecture

2

Installation Guidelines

This chapter explains how to install an ADAM-4500 Series Controller. A quick hookup scheme is provided that let you easily configure your system before implement it into your application.

2.1 System Requirements

Before you start to install the ADAM-4500 Series Controller, make sure the system requirements as below:

2.1.1 Host Computer Requirements

1. IBM PC compatible computer with 486 CPU (Pentium grade is recommended).
2. Microsoft XP/2000/NT/98/95 or higher versions.
3. DOS version 3.31 or higher.
3. Borland C 3.0 for DOS.
4. At least 32 MB RAM.
5. 20 MB of hard disk space available
6. VGA color monitor.
7. 2x or higher speed CD-ROM.
8. Mouse or other pointing devices.
9. At least one standard RS-232 port (e.g. COM1, COM2).
10. One RS-485 card or RS-232 to RS-485 converter (e.g. ADAM-4520) for system communication.
11. Hub or Switch for Ethernet connection.

2.2 Hardware Installation

2.2.1 Jumper Settings

This section tells you how to set the jumpers to configure your ADAM-4500 Series Controller. It provides system default configuration and your options for each jumper. There are four jumpers on the CPU card:

JP2 is for the reset mode setting.
JP5 is for battery backup RAM setting.
JP6 & **JP7** is for COM4 communication mode selection setting.

The following figure shows the location of the jumpers:

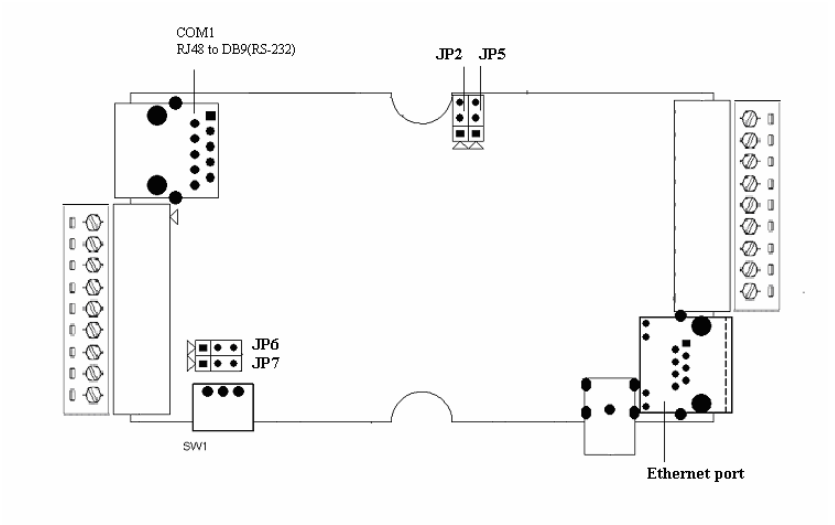


Figure 2-1 ADAM-4500 Series Jumper Definition

Reset Mode Setting

Jumper JP2 on the CPU card lets you configure the reset mode:

1. **Watchdog timer reset mode:** if watchdog timer is enable, it have to be cleared at least once every 1.6 seconds, or the system will reboot. Please refer section 5.3.1 to see how to use function library to configure watchdog timer.
2. **NMI (Non-maskable interrupt) reset mode:** once the NMI is activated, the system will reboot.
3. **Non-reset mode:** there is no reset mode set for system.

The default setting of watchdog timer is “watchdog timer reset mode“. Jumper settings are shown below:

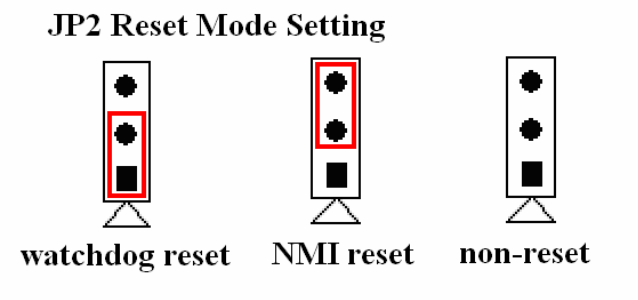


Figure 2-2 Jumper of Reset Mode

Battery Backup RAM Setting

Jumper JP5 on the CPU card lets you configure the battery backup for SRAM is Enable or Disable. The default setting of battery backup is “Enable”. Jumper settings are shown below:

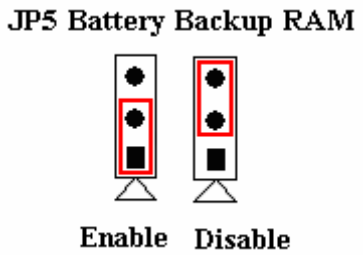


Figure 2-3 Jumper of battery backup

COM4 Communication Mode Selection Setting

The communication mode of COM4 is setting by the Jumper JP6 and JP7. Please refer to Figure 2-9 to set the communication mode. The default setting of COM4 is Programming port (RS-232 mode) for program download. You can also set COM4 as RS-485 mode. Jumper settings are listed below:

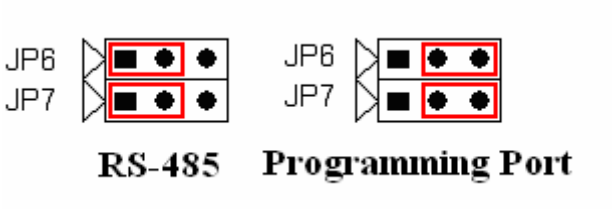


Figure 2-4 Jumper of COM4 communication mode selection

2.3 System Wiring and Connections

This section provides basic information on wiring the power supply, I/O units, communication port connection and programming port connection.

2.3.1 Power Supply Wiring

Although the ADAM-4500 Series Controller is designed for using a standard industrial unregulated 24 V DC power supply, they accept any power unit that supplies within the range of +10 to +30 VDC . The power supply ripple must be limited to 200 mV peak-to-peak, and the immediate ripple voltage should be maintained between +10 and +30
V_{DC}. Screw terminals +Vs and GND are for power supply wiring and the wires used should be sized at least 2 mm.

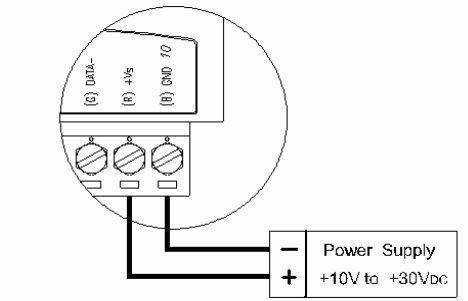


Figure 2-5 ADAM-4500 Series Controller power wiring

2.3.2 I/O modules wiring

The system uses a plug-in screw terminal block for the interface between I/O modules and field devices. The following information must be considered:

1. The terminal block accepts wires from 0.5 mm to 2.5 mm.
2. Always use a continuous length of wire. Do not combine wires to make them longer.
3. Use the shortest possible wire length.
4. Use wire trays for routing where possible.
5. Avoid running wires near high energy wiring.
6. Avoid running input wiring in close proximity to output wiring where possible.

The figures below show how to connect external signals to I/O
connector of ADAM-4500 Series Controller.

2.3.1.1 ADAM-4501/4501D

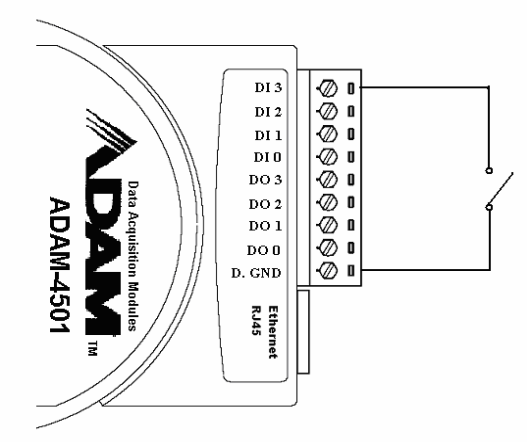


Figure 2-6 Dry contact wiring for DI channel

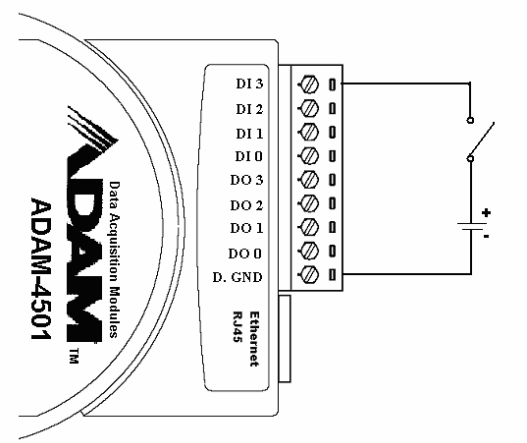


Figure 2-7 Wet contact wiring for DI channel

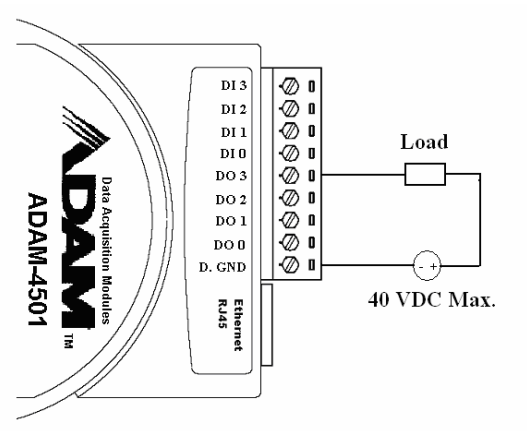
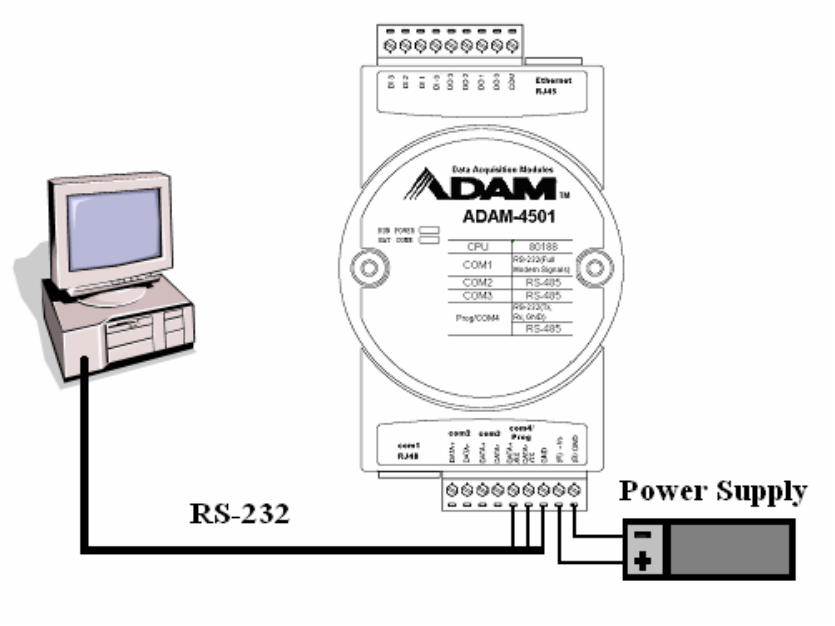


Figure 2-8 Digital output wiring

2.3.3 System Network Connection

Network Connection for System Configuration and Download

The ADAM-4500 Series Controller has a Programming port. This port (COM4) allows you to program, configure, and troubleshoot the ADAM-4500 Series Controller from your host computer. The Programming port has an RS-232 interface and only uses TX, RX, and GND signals. The cable connection and the pin assignment are as follows:



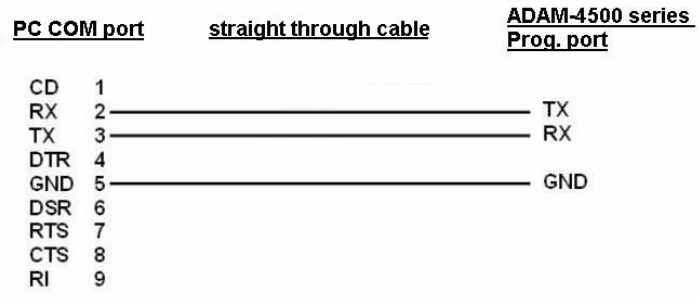
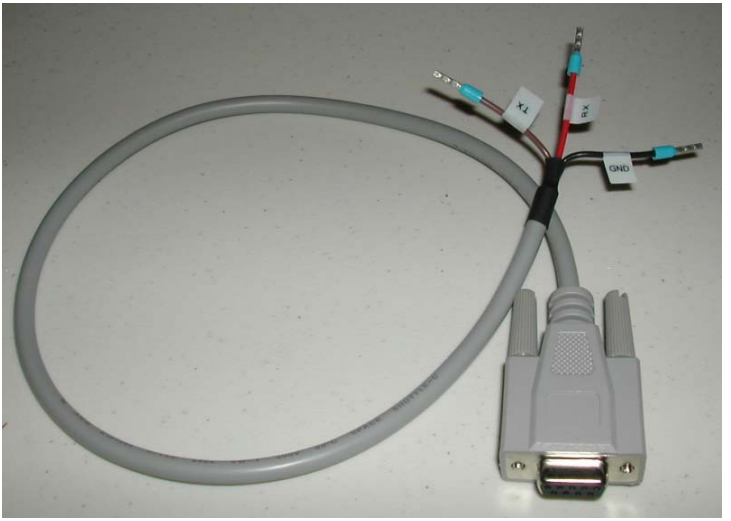


Figure 2-9 System Configuration Wiring

Note: We also provide a friendly cable for Programming port wiring. Please refer following pictures to wire the Programming port.



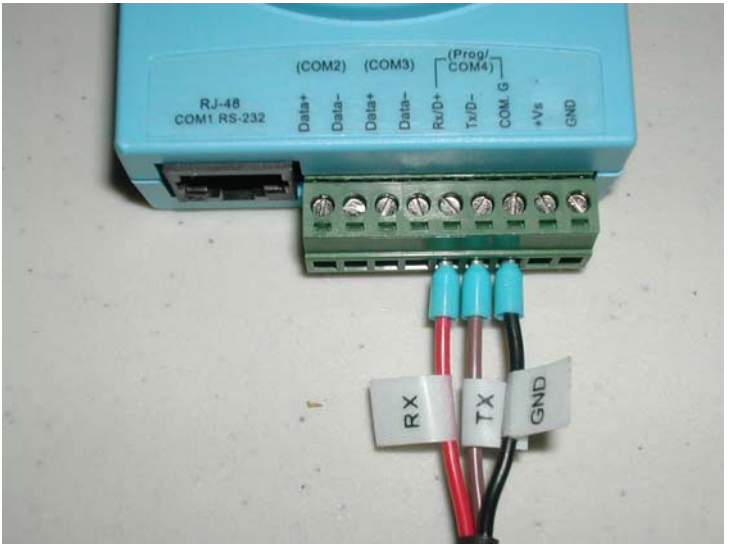


Figure 2-10 Cable Configuration Wiring

RS-232 Network Connection for System Monitoring and Integration

Since the connection for an RS-232 interface is not standardized, different devices implement the RS-232 connection in different ways.
If you are having problems with a serial device, be sure to check the pin assignments for the connector. We provide one cable to transfer RJ-48 (COM1 connector on ADAM-4500 Series Controller) to DB9.
The following figure shows the pin assignments of the DB9 connector for the ADAM-4500 Series Controller COM1 RS-232 port.

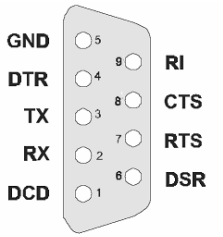


Figure 2-11 Pin Assignment of COM1

RS-485 Network Connection for System Monitoring and Integration

The ADAM-4500 Series Controller provides RS-485 interfaces for multi-drop network integration. The COM2 & COM3 are dedicated RS-485 interface. The COM4 is an RS-232/485 selectable COM port. They are able to support the Modbus/RTU master and slave functions.

Ethernet Network Connection

The ADAM-4500 Series Controller provides Ethernet interface (RJ-45 type) for network integration. Usually, you will need to prepare an Ethernet switch like as ADAM-6520 or hub for connecting to other network devices as following figure.

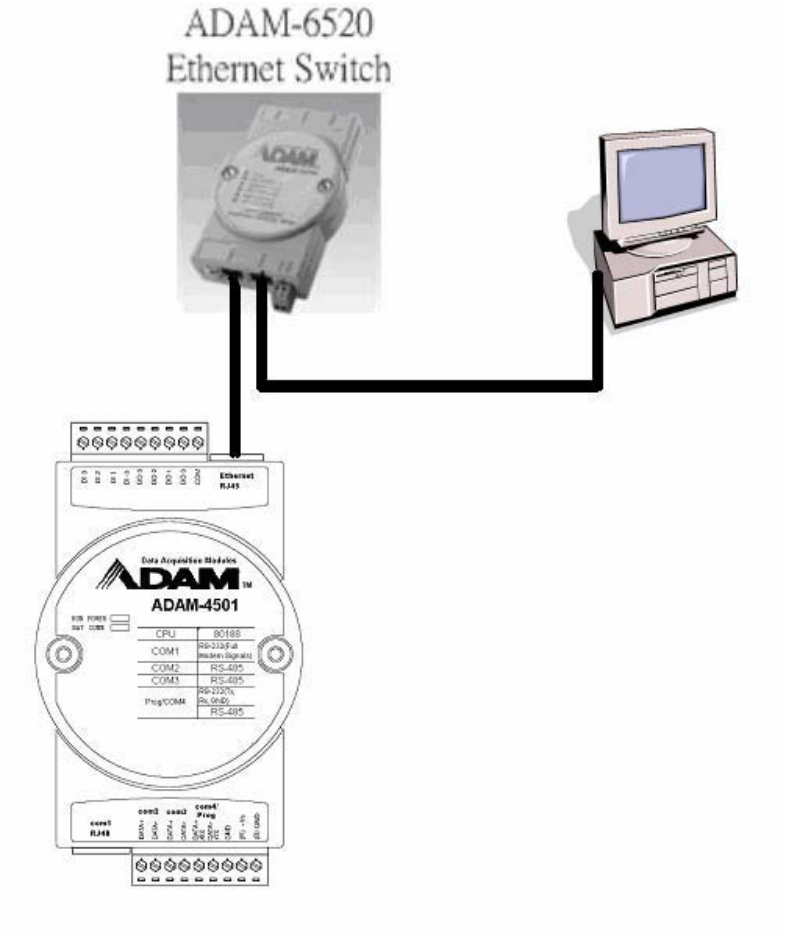


Figure 2-12 Ethernet Connection

2.4 Software Installation

ADAM-4500 Series Controller comes with a Utility CD, containing ADAM Product Series Utilities as system configuration tool. While you Insert the CD into the CD drive (e.g. D:) of the host PC, the Utility software setup menu will start up automatically. Click the ADAM-4500
Series icon to execute the setup program, and there will be a Utility executive program installed in your host PC. See **Chapter 3 I/O Configuration and Program Download** for the detail operation.

3

Program Download

This chapter explains how to use the ADAM-4500 Series Utility to
configure and download application programs

3.1 System Hardware Configuration

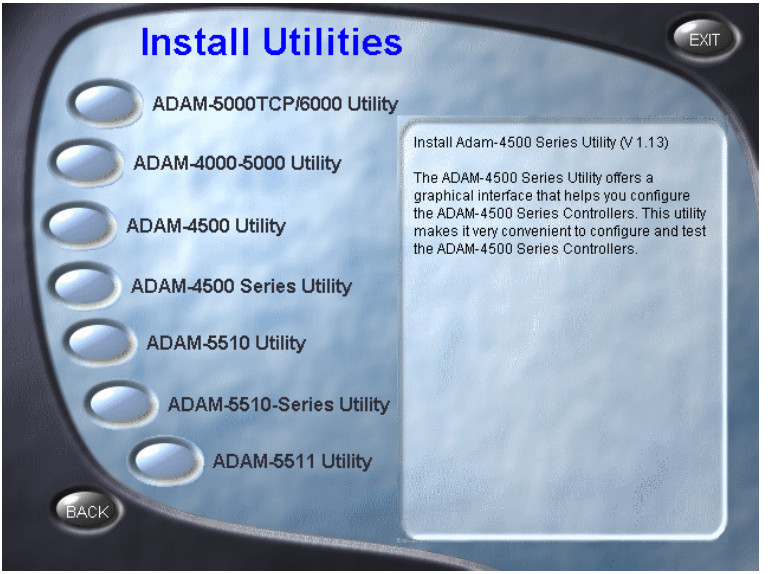
Before the system configuration, you will need to setup the environment as mentioned in Chapter 2.1: System Requirements.

3.2 Install Utility Software on Host PC

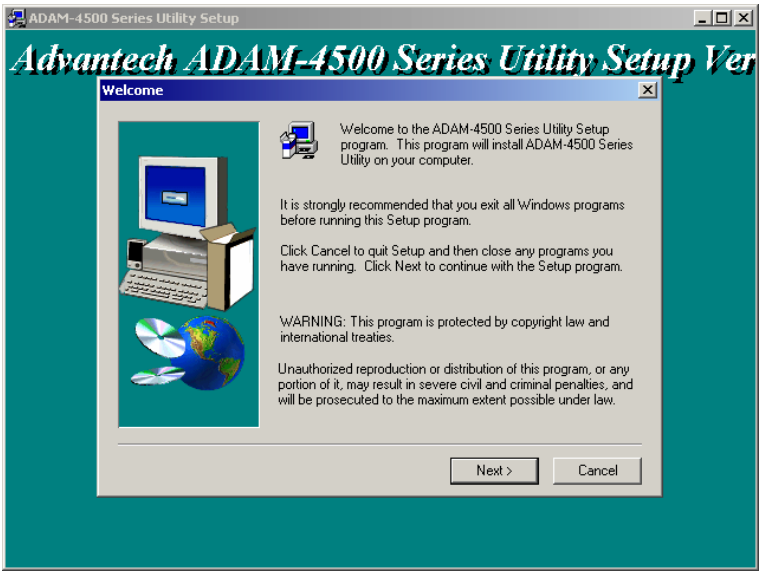
ADAM-4500 Series Controller comes with a Utility CD, containing
ADAM Product Series Utilities as system configuration tools.

1. Insert the ADAM Utility CD into the CD drive of the host PC, and the Utility software setup menu will start automatically. Click the “Install Utility” button to install ADAM Series Utility.

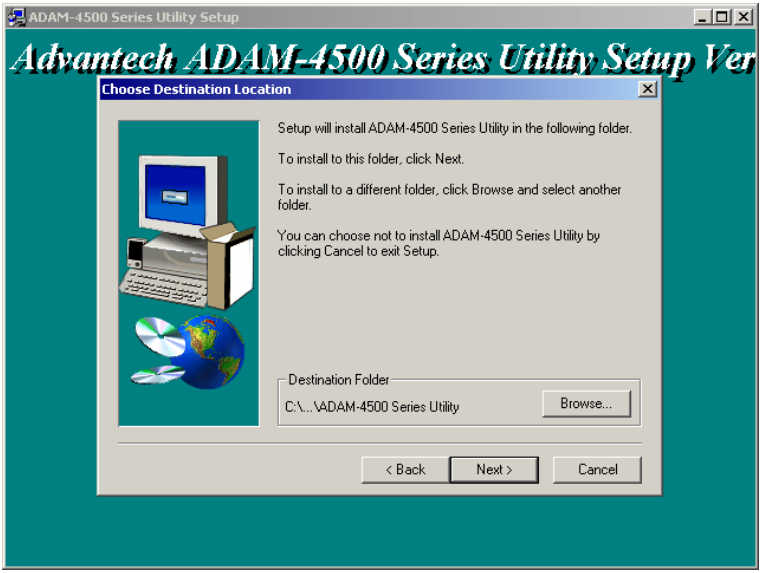
2. Click on the “ADAM-4500 Series Utility” to setup ADAM-4500
Series Utility.


Install Utilities
EXIT
ADAM-5000TCP/6000 Utility
ADAM-4000-5000 Utility
ADAM-4500 Utility
ADAM-4500 Series Utility
ADAM-5510 Utility
ADAM-5510-Series Utility
ADAM-5511 Utility
BACK
Install Adam-4500 Series Utility (V 1.13)
The ADAM-4500 Series Utility offers a graphical interface that helps you configure the ADAM-4500 Series Controllers. This utility makes it very convenient to configure and test the ADAM-4500 Series Controllers.

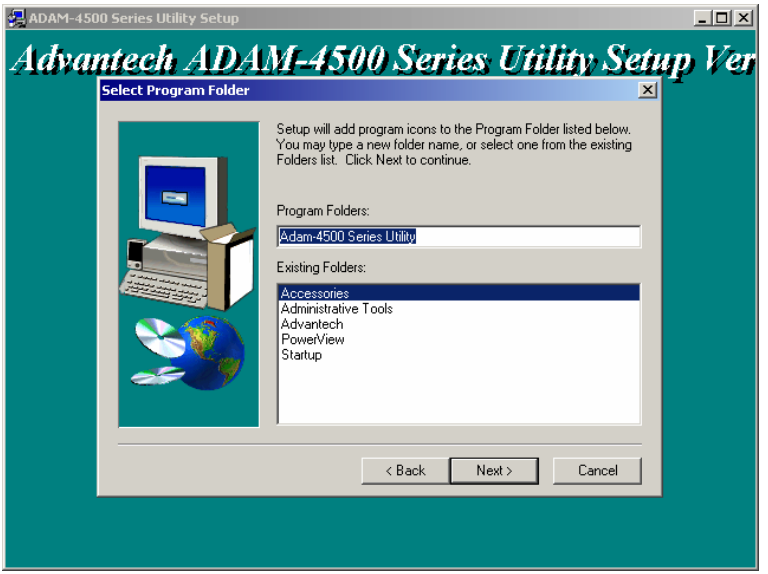
3. There will be dialog window showing, and click the ‘Next” button.


ADAM-4500 Series Utility Setup
Advantech ADAM-4500 Series Utility Setup Ver
Welcome
Welcome to the ADAM-4500 Series Utility Setup program. This program will install ADAM-4500 Series Utility on your computer.
It is strongly recommended that you exit all Windows programs before running this Setup program.
Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.
WARNING: This program is protected by copyright law and international treaties.
Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.
Next >
Cancel

4. Choose the dictionary path for installing ADAM-4500 Series Utility, and then click the “Next” button.

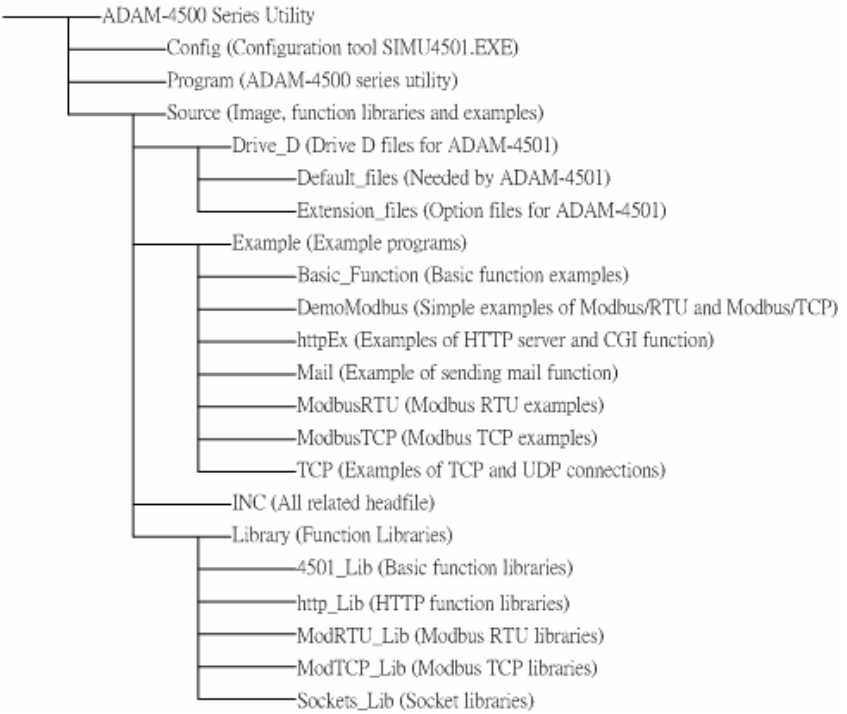

ADAM-4500 Series Utility Setup
Advantech ADAM-4500 Series Utility Setup Ver
Choose Destination Location
Setup will install ADAM-4500 Series Utility in the following folder.
To install to this folder, click Next.
To install to a different folder, click Browse and select another folder.
You can choose not to install ADAM-4500 Series Utility by clicking Cancel to exit Setup.
Destination Folder
C:\...\ADAM-4500 Series Utility
Browse...
< Back
Next >
Cancel

5. Set name for Program Folder and click the “Next” button.


ADAM-4500 Series Utility Setup
Advantech ADAM-4500 Series Utility Setup Ver
Select Program Folder
Setup will add program icons to the Program Folder listed below. You may type a new folder name, or select one from the existing Folders list. Click Next to continue.
Program Folders:
Adam-4500 Series Utility
Existing Folders:
Accessories
Administrative Tools
Advantech
PowerView
Startup
< Back
Next >
Cancel

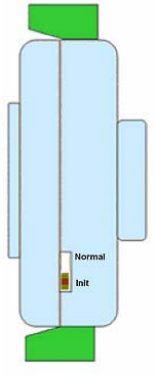
6. Click the “Finish” button to complete the installation.

7. After the ADAM-4500 Series Utility has been installed, you will find three directories under “C:\Program Files\Advantech\ADAM-4500 Series Utility” directory as following:

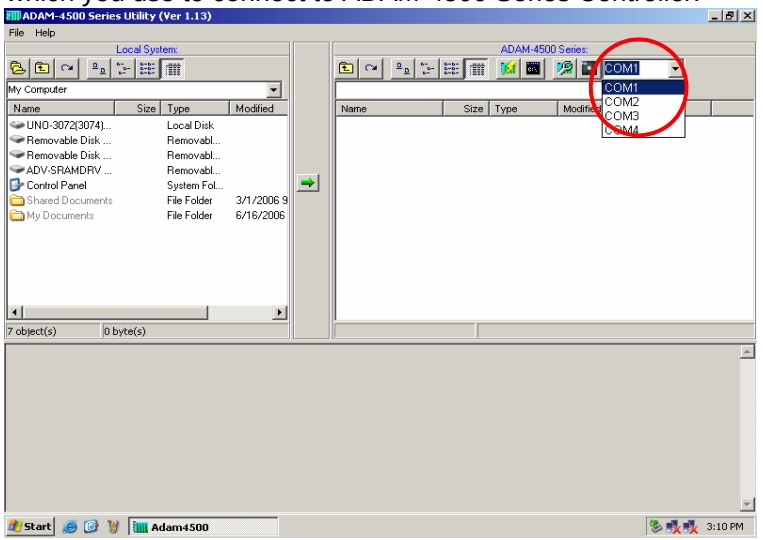

ADAM-4500 Series Utility
Config (Configuration tool SIMU4501.EXE)
Program (ADAM-4500 series utility)
Source (Image, function libraries and examples)
Drive_D (Drive D files for ADAM-4501)
Default_files (Needed by ADAM-4501)
Extension_files (Option files for ADAM-4501)
Example (Example programs)
Basic_Function (Basic function examples)
DemoModbus (Simple examples of Modbus/RTU and Modbus/TCP)
httpEx (Examples of HTTP server and CGI function)
Mail (Example of sending mail function)
ModbusRTU (Modbus RTU examples)
ModbusTCP (Modbus TCP examples)
TCP (Examples of TCP and UDP connections)
INC (All related headfile)
Library (Function Libraries)
4501_Lib (Basic function libraries)
http_Lib (HTTP function libraries)
ModRTU_Lib (Modbus RTU libraries)
ModTCP_Lib (Modbus TCP libraries)
Sockets_Lib (Socket libraries)

3.3 ADAM-4500 Series Utility Overview

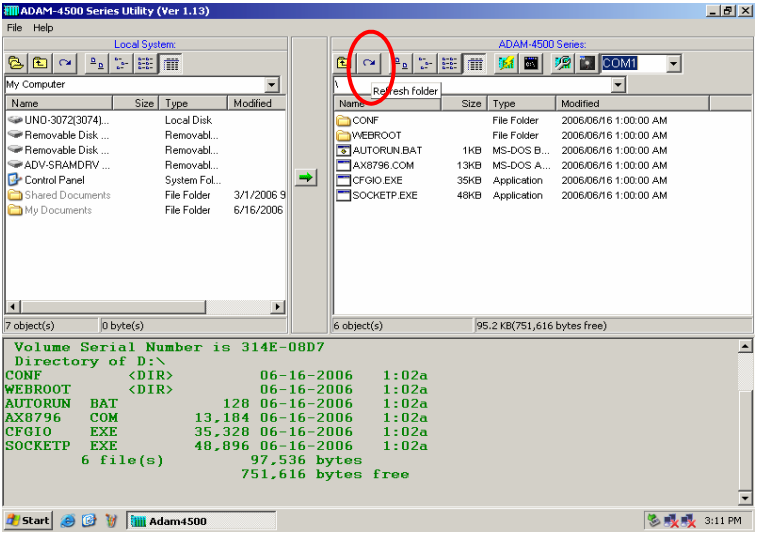
1. Put switch into **Initial mode** and then reboot. (Initial mode is for configuration and download program into ADAM-4501 controller. Always set **Initial mode** when downloading program. **Normal mode** will automatically run the downloaded program in the autorun.bat file when ADAM-4500 Series Controller boots up. Refer to section 3.6 for more detail)



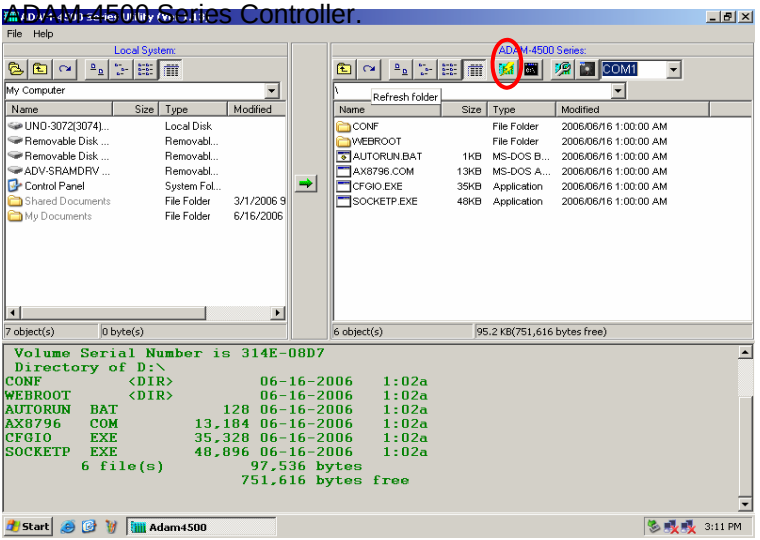
2. Execute Adam4500.exe under C:\Program Files\Advantech\ ADAM-4500 Series Utility\Program. Select the COM Port on your PC
which you use to connect to ADAM-4500 Series Controller.



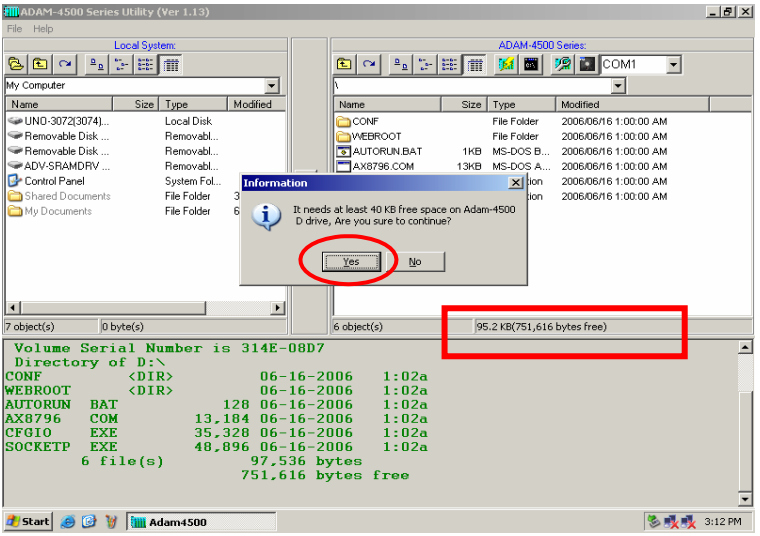
3. Click the “Refresh Folder” button to display the files and directories on the drive D of ADAM-4500 Series Controller.



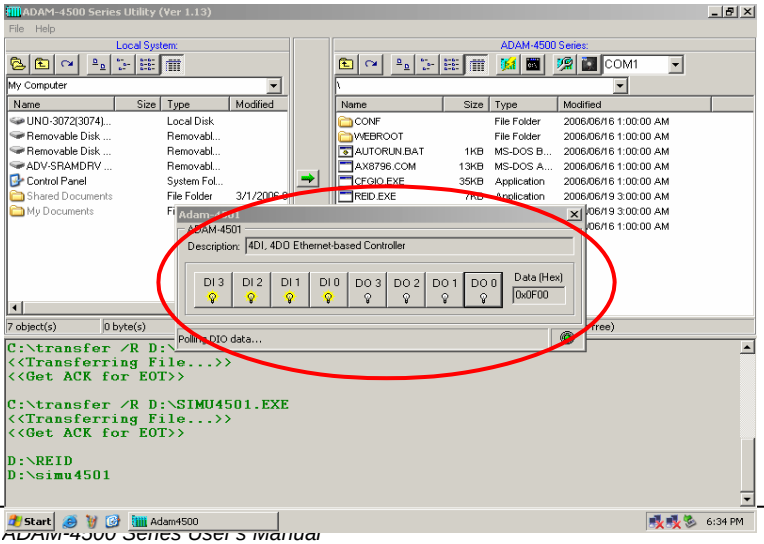
4. Click the “Config ADAM” button to test digital input/output on ADAM-4500 Series Controller.



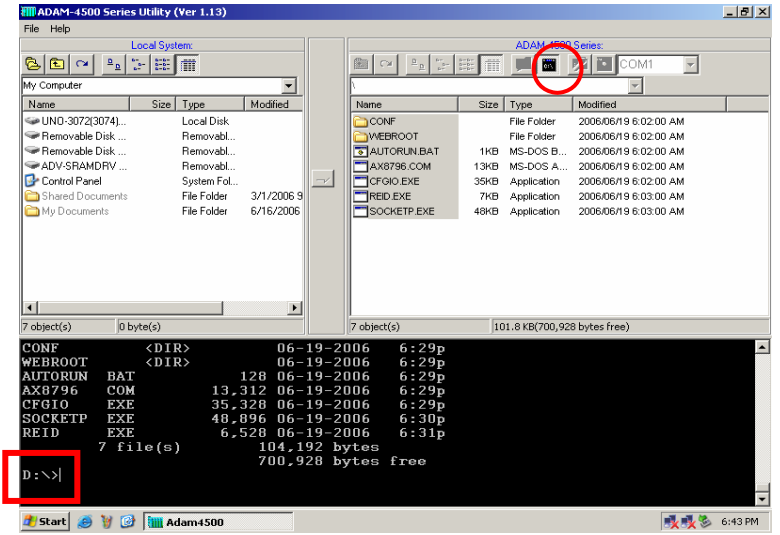
5. Check if there is enough space for download this application to disk on ADAM-4500 Series Controller (Refer to the status bar). If the space is greater than 40KB, click the “Yes” button to continue.


ADAM-4500 Series Utility (Ver 1.13)
Information
It needs at least 40 KB free space on Adam-4500 D drive, Are you sure to continue?
Yes
No
95.2 KB(751,616 bytes free)
Volume Serial Number is 314E-08D7
Directory of D:\
CONF <DIR> 06-16-2006 1:02a
WEBROOT <DIR> 06-16-2006 1:02a
AUTORUN BAT 128 06-16-2006 1:02a
AX8796 COM 13,184 06-16-2006 1:02a
CFGIO EXE 35,328 06-16-2006 1:02a
SOCKETP EXE 48,896 06-16-2006 1:02a
6 file(s) 97,536 bytes
751,616 bytes free

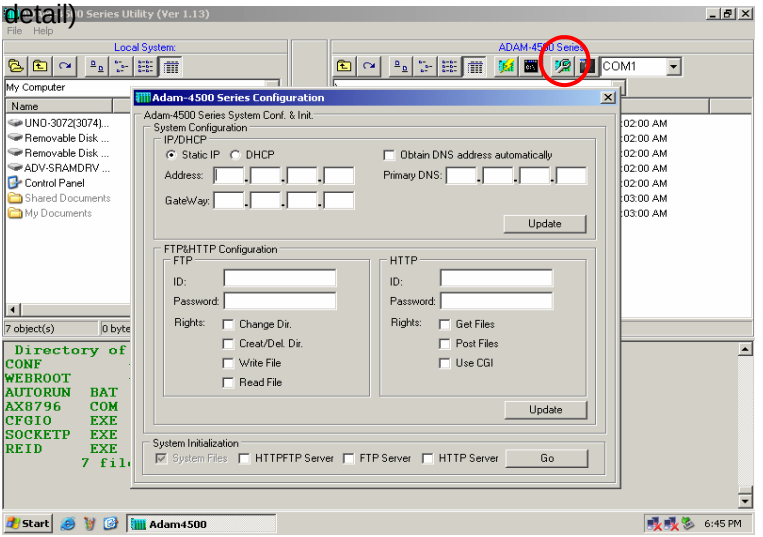
6. Built-in example will be downloaded into the controller. You can use this example to test the functionality of DIO on ADAM-4501/4501D.



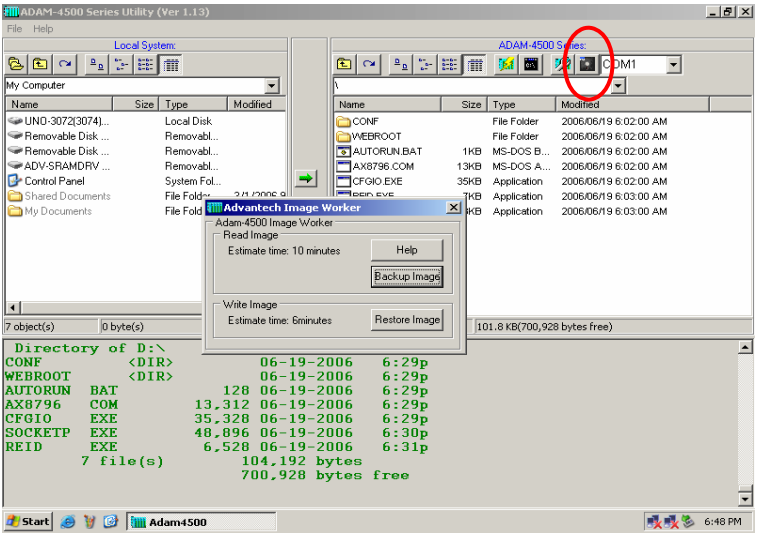
7. Click the “Launch Terminal” button for launching terminal emulation function. You can type DOS command in the termination window to communicate with ADAM-4500 Series Controller.



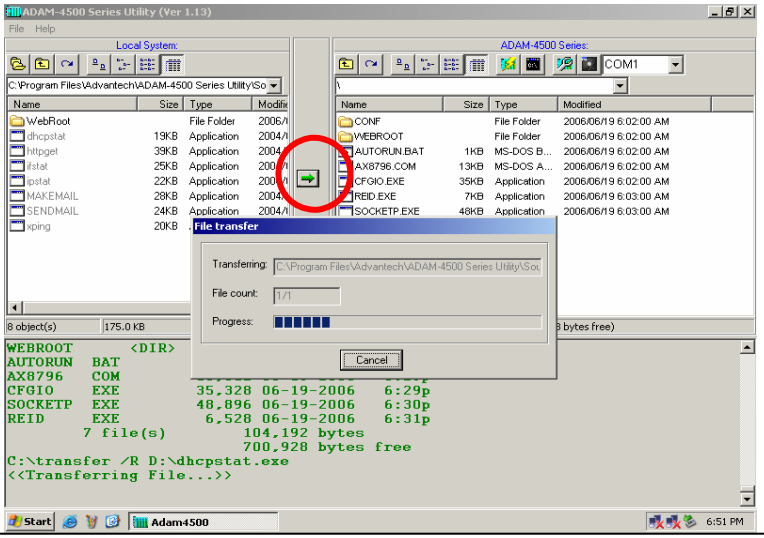
8. Click the “Adam-4500 Configuration” button for configuring network (such as IP address, Gateway and DNS), FTP/HTTP server settings and performing system initialization function. (Refer to section 3.4 and 3.5 for more detail)



9. Click the “Advantech Image Worker” button for backup drive D as image file or restore image file to drive D. (Refer to section 3.7 and 3.8 for more detail)



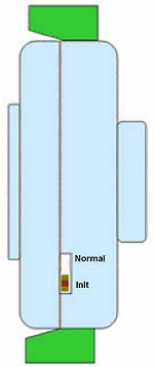
10. Click the “Download” button for downloading programs or files from host PC (on the left window) to ADAM-4500 Series Controller (on the right window).



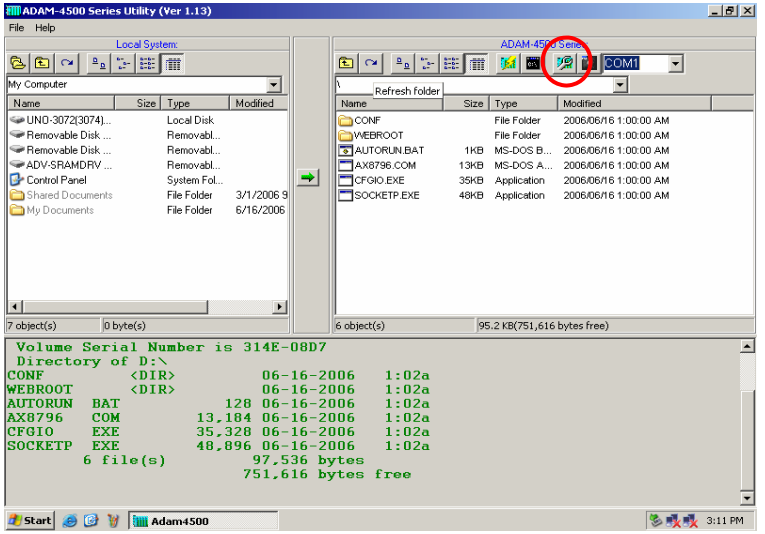
3.4 Initialize the drive D: to default settings.

Following steps will show you how to initialize the drive D to default settings for ADAM-4500 Series Controller. The drive D of ADAM-4500 Series Controller will return to initial files and settings after this function is performed.

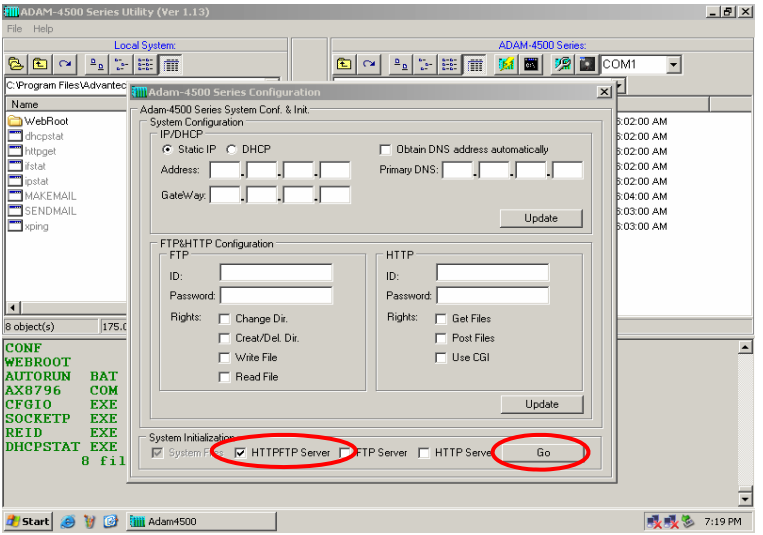
1. Put switch into **Initial mode** and then reboot.



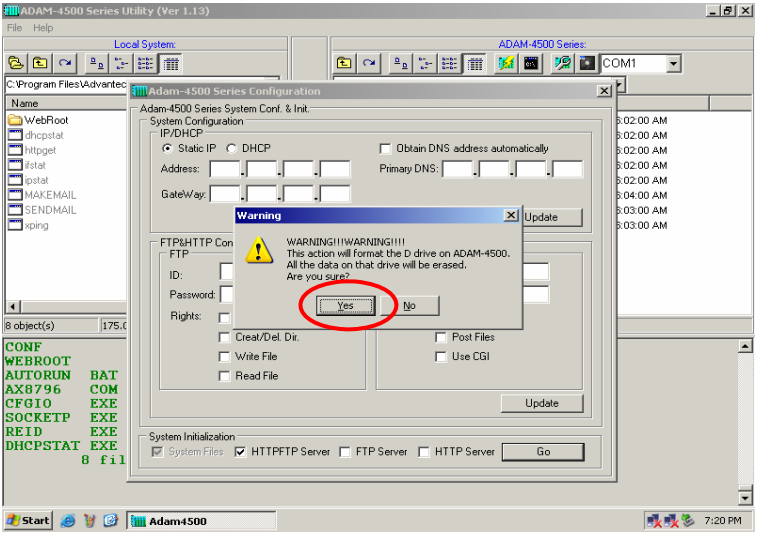
2. Click the “Adam-4500 Configuration” button.



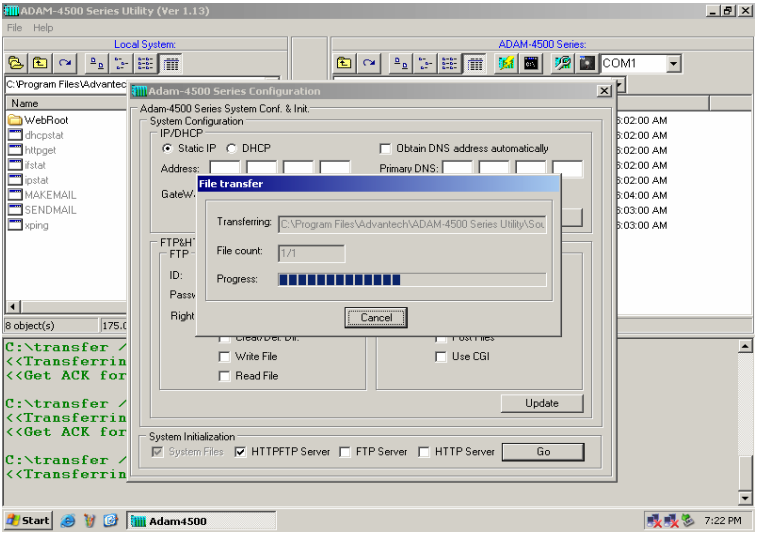
3. Select “HTTPFTP Server” and click the “Go” button. (This means both HTTP and FTP server will be initialized)



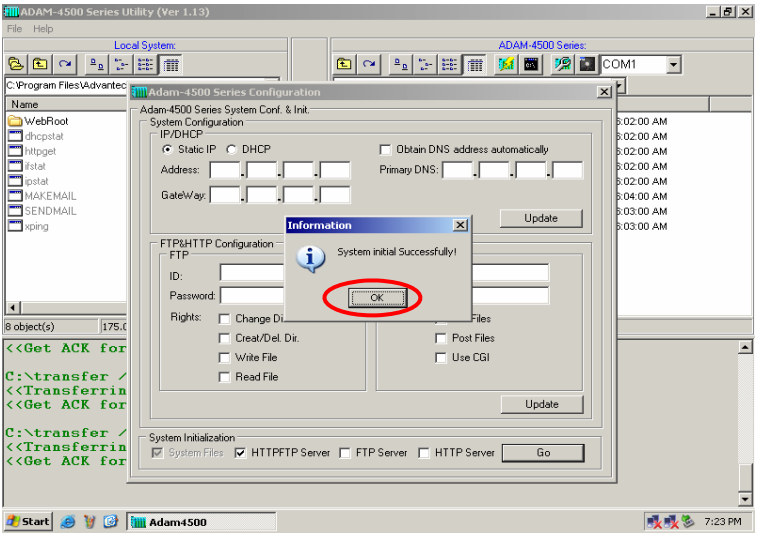
4. There will be window showing “Warning”, click “Yes” to initialize drive D. Please note that drive D will be formatted and all the files on drive D will be lost. If you would like to backup the drive contents, please refer to section 3.7.



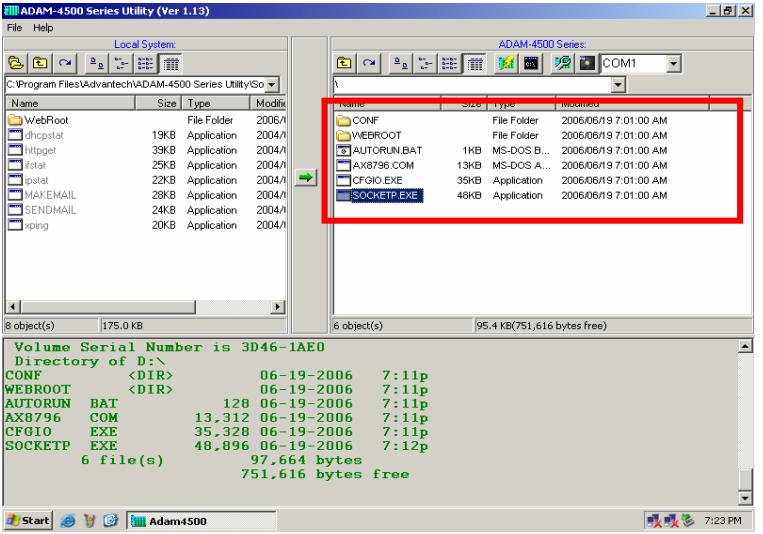
5. You will find the initialization process is performing.



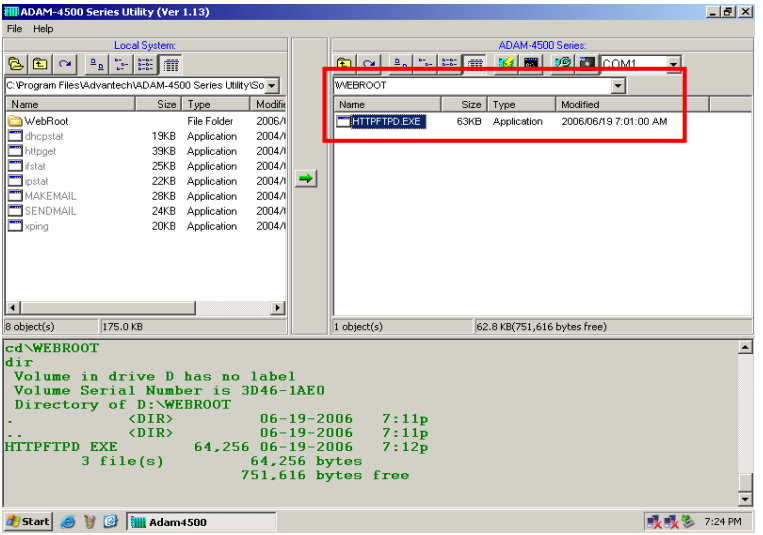
6. Click the “OK” button to finish the initialization process.



7. The directory of drive D will be refreshed as following picture.



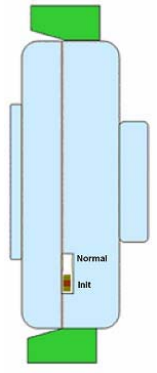
8. The FTP and HTTP Server file is under “WEBROOT” directory.



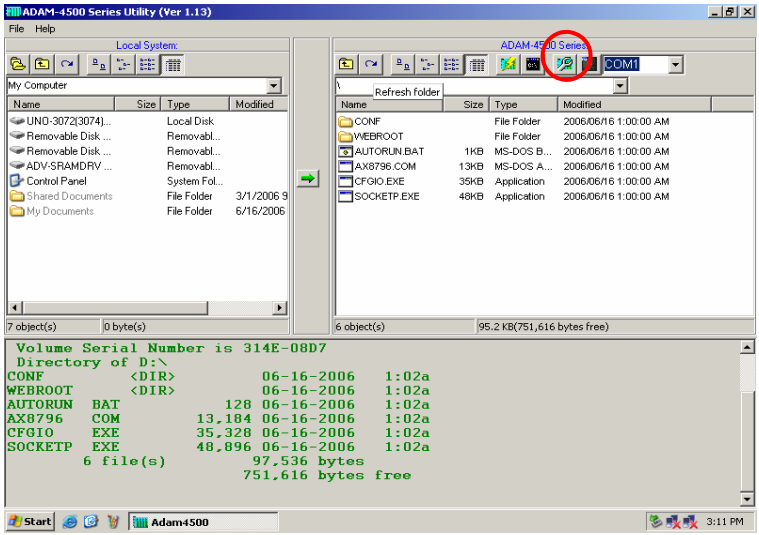
3.5 Configure IP address and ftp/http user/password settings.

Following steps will show you how to configure IP address and users/password of FTP server and HTTP server for ADAM-4500
Series Controller. *Please note the default IP address is "10.0.0.1".*

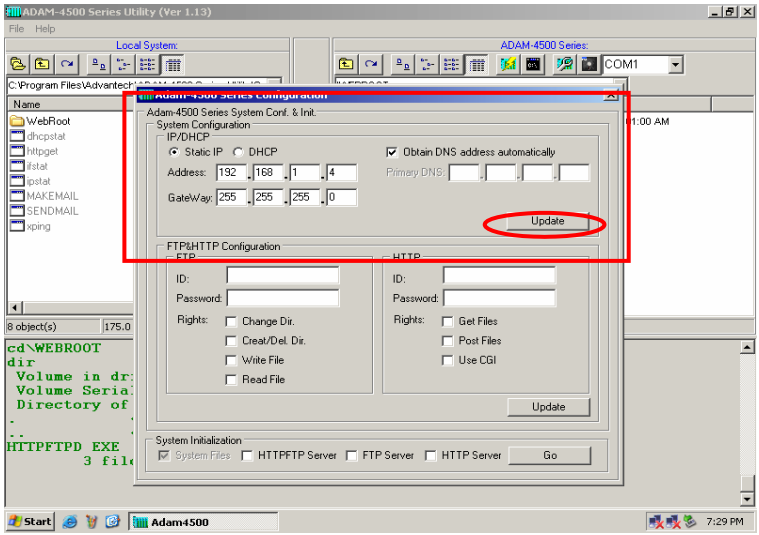
1. Put switch into **Initial mode** and then reboot.



2. Click the "Adam-4500 Configuration" button.

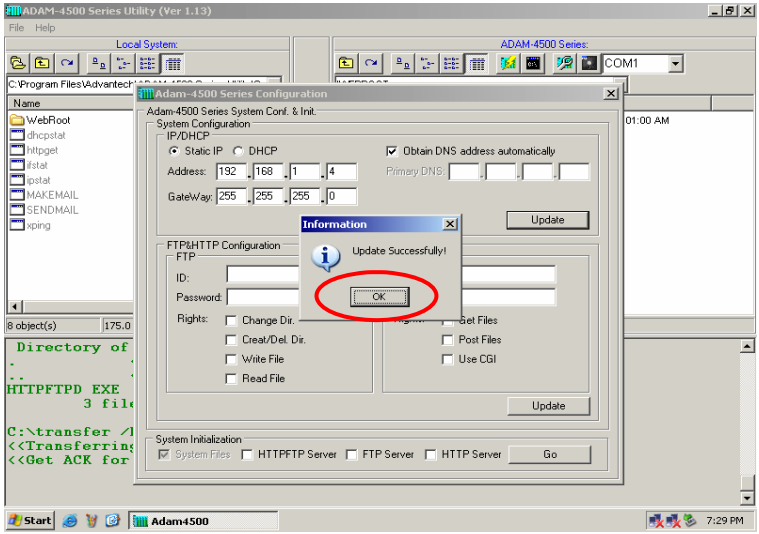


3. Select “Static IP” and fill in the IP address and Gateway IP. Select “Obtain DNS address automatically”. Click the “Update” button to perform the configuration.

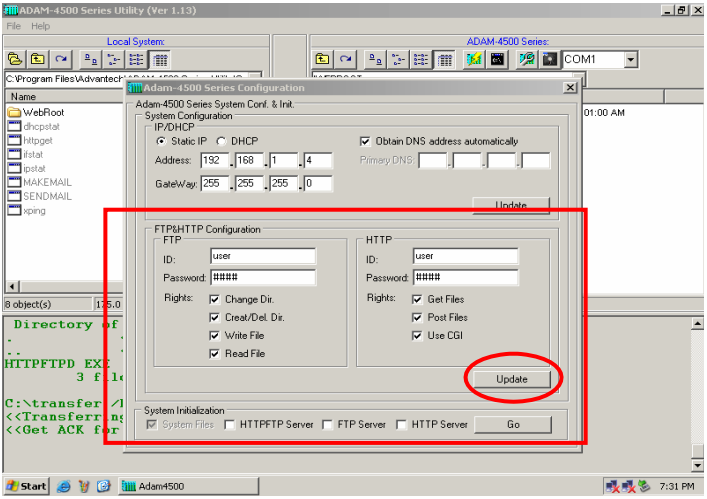


Note: Above settings is only an example. You have to configure the network settings according to your network environment.

4. Click the “OK” button to finish the network configuration.

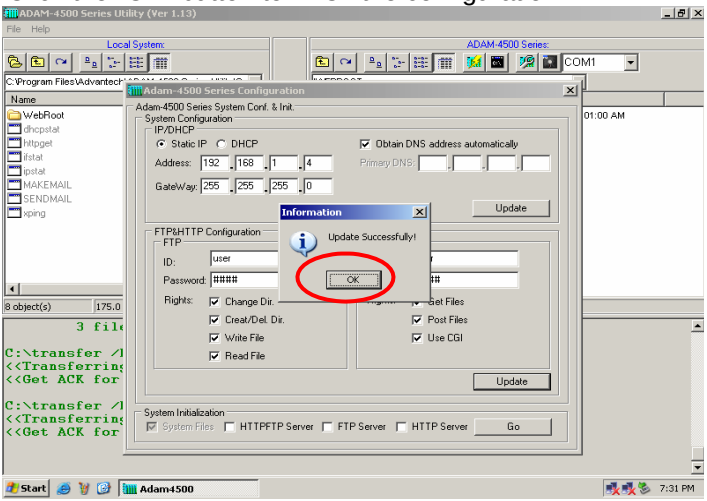


5. Fill in the user name, password and access right for FTP server and HTTP server. Click the "Update" button to perform the configuration.



***Note*:** This utility can only let you configure one user for FTP server and one user for HTTP server. If you would like to configure multi- users for FTP server and HTTP server, please refer to chapter 4.

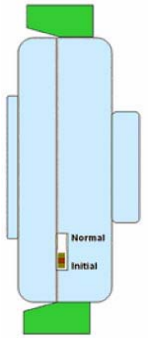
6. Click the "OK" button to finish the configuration.



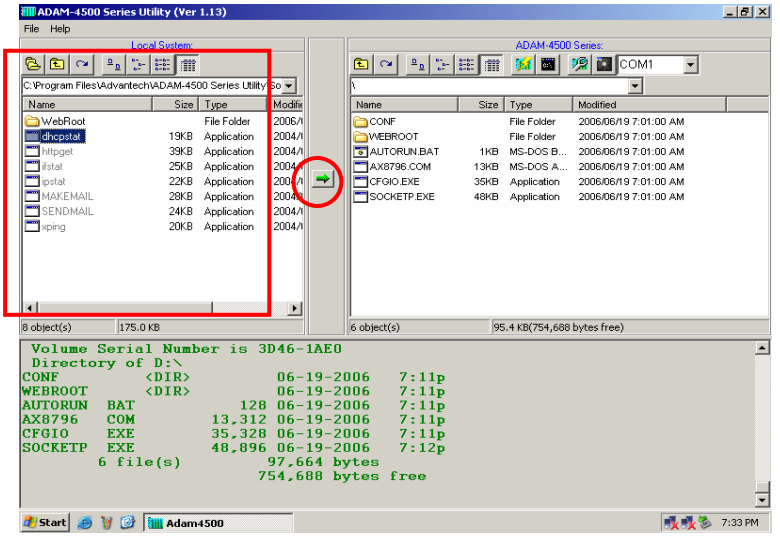
3.6 Download and run the application program automatically after boot up

ADAM-4500 series can automatically run program after boot up by including the program in the autorun.bat file. Following steps will demonstrate how to run dhcpstat.exe automatically after boot up.

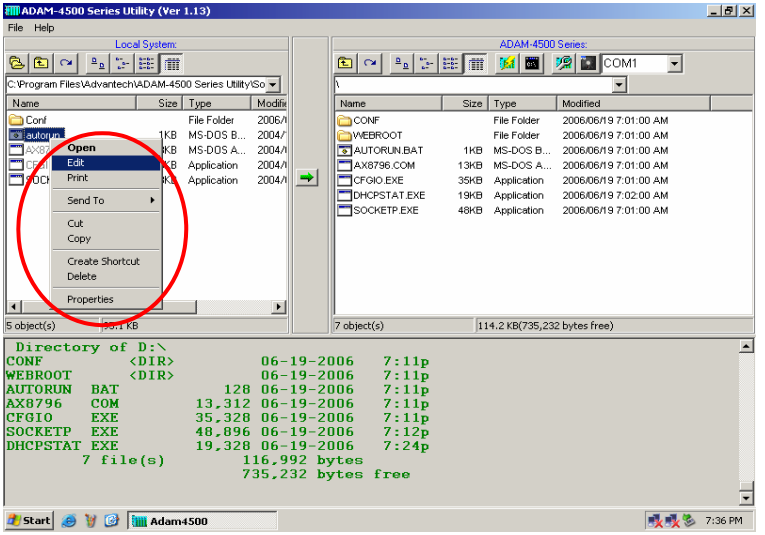
1. Put switch into **Initial mode** and then reboot.



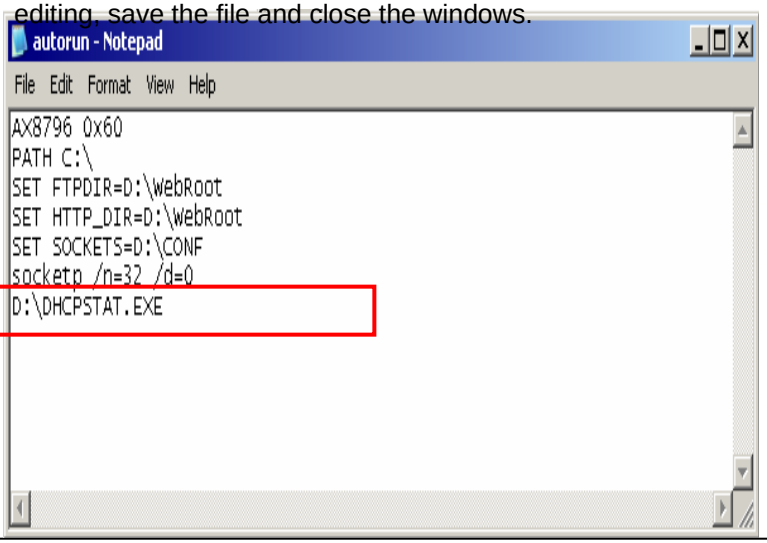
2. Download the program dhcpstat.exe (under C:\Program Files\ Advantech\ADAM-4500 Series Utility\Source\Drive_D\Extension_ files) onto ADAM-4500 Series Controller by click the “Download” button.



3. Find the autorun.bat file under C:\Program Files\Advantech\
ADAM-4500 Series Utility\Source\Drive_D\Default_files on the left window (on the Host PC). Right click the autorun.bat and choose “Edit”.

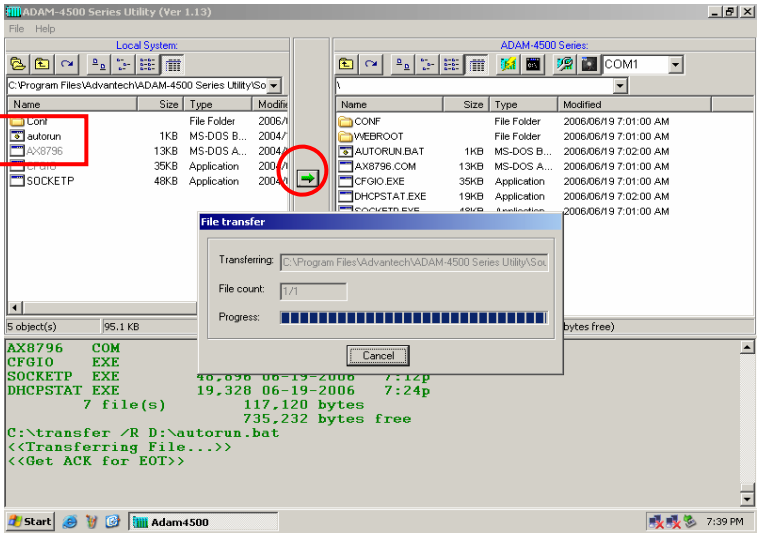


4. Add “D:\DHCPSTAT.EXE” in the autorun.bat file. After finishing
editing, save the file and close the windows.



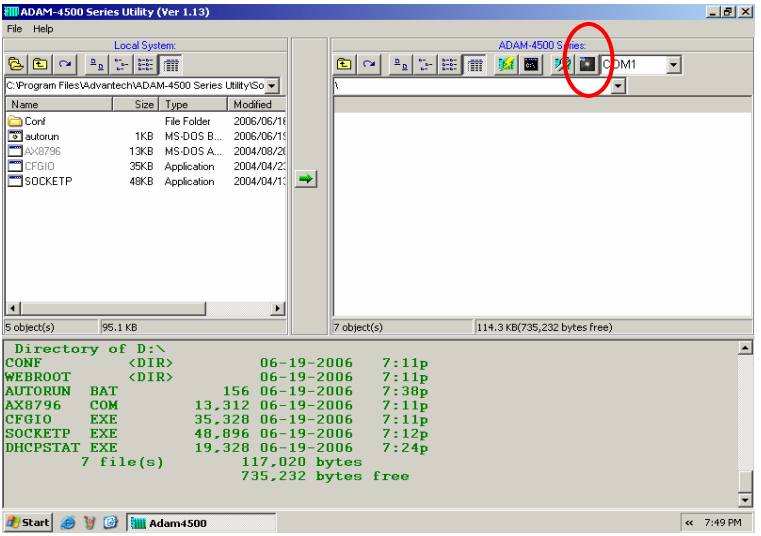
5. Update modified autorun.bat file onto ADAM-4500 Series

Controller by choose the autorun.bat file in the left window and click the “Download” button.

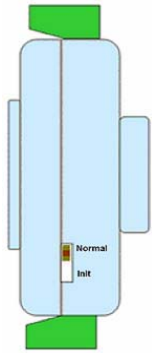


6. Click the “Launch Terminal” button for launching terminal

emulation function.

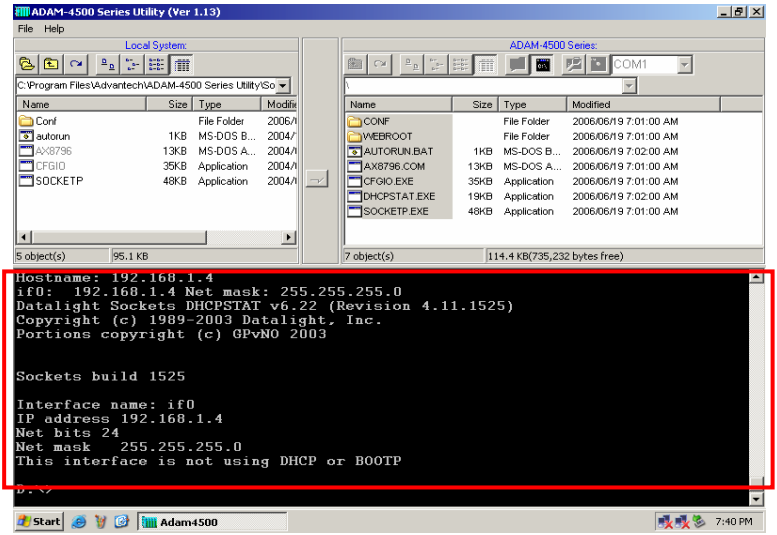


7. Put switch into **Normal mode** and then reboot. (In **Initial mode**,
program listed in autorun.bat will not automatically run after boot up. it will automatically run after boot up only in **Normal mode**)



.

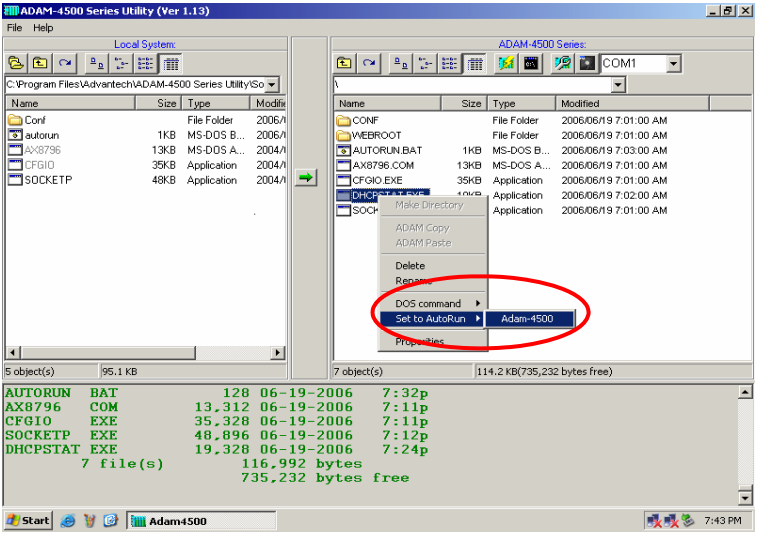
8. Reboot the ADAM-4500 Series Controller and check the terminal window to see if dhcpstat.exe has been executed correctly.


ADAM-4500 Series Utility (Ver 1.13)
File Help
Local System:
ADAM-4500 Series:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type Modifie
Conf File Folder 2006/
autorun 1KB MS-DOS B... 2004/
AX8796 13KB MS-DOS A... 2004/
CFGIO 35KB Application 2004/
SOCKETP 48KB Application 2004/
5 object(s) 95.1 KB
COM1
Name Size Type Modified
CONF File Folder 2006/06/19 7:01:00 AM
WEBROOT File Folder 2006/06/19 7:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/06/19 7:02:00 AM
AX8796.COM 13KB MS-DOS A... 2006/06/19 7:01:00 AM
CFGIO.EXE 35KB Application 2006/06/19 7:01:00 AM
DHCPSTAT.EXE 19KB Application 2006/06/19 7:02:00 AM
SOCKETP.EXE 48KB Application 2006/06/19 7:01:00 AM
7 object(s) 114.4 KB(735,232 bytes free)
Hostname: 192.168.1.4
if0: 192.168.1.4 Net mask: 255.255.255.0
Datalight Sockets DHCPSTAT v6.22 (Revision 4.11.1525)
Copyright (c) 1989-2003 Datalight, Inc.
Portions copyright (c) GPvNO 2003
Sockets build 1525
Interface name: if0
IP address 192.168.1.4
Net bits 24
Net mask 255.255.255.0
This interface is not using DHCP or BOOTP
D:\>
Start
Adam4500
7:40 PM

Note: There is another easier way to set program run automatically after boot up. You don't need to modify the autorun.bat on the host PC and download to ADAM-4500 Series Controller.

In this example, put the switch into **Initial Mode**. Download the program dhcpstat.exe onto ADAM-4500 Series Controller (as shown
in step 2 of the previous method).

Then right click the program dhcpstat.exe on the right window and choose "Set to AutoRun>>Adam-4500". It will modify the autorun.bat (this has the same effect as step 3~5 of the previous method)

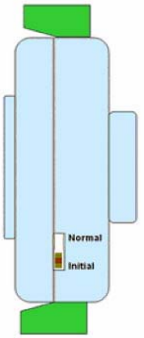


After that, click the "Launch Terminal" button (as shown by steps 6 of previous method). Put the switch into **Normal mode** and then reboot. Now you should see the terminal window the same as steps 8 of previous method.

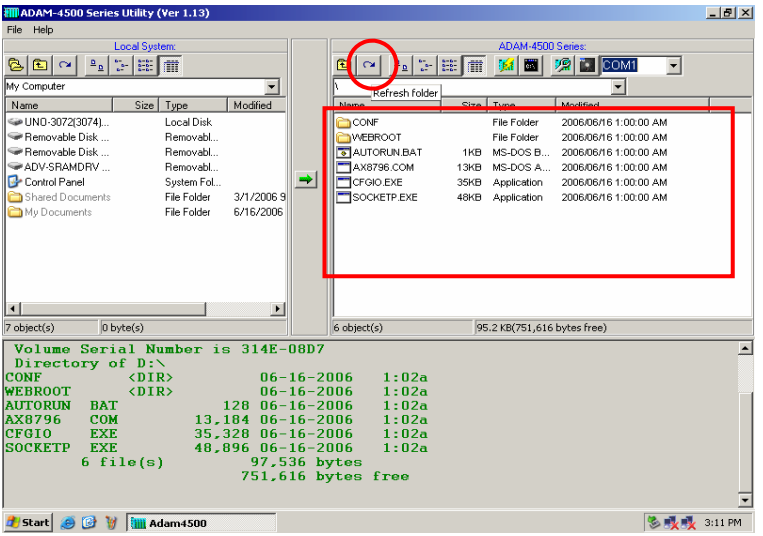
3.7 Backup drive D as image file.

Following steps demonstrate how to backup drive D as image file.

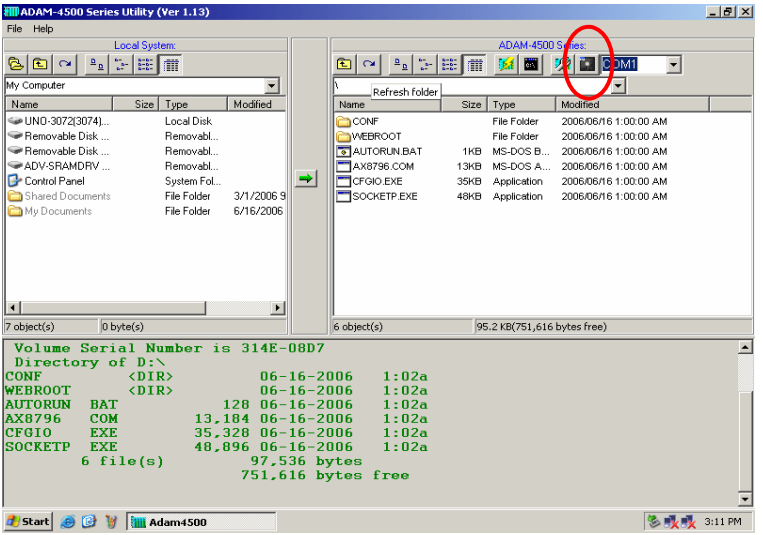
1. Put switch into **Initial mode** and then reboot.



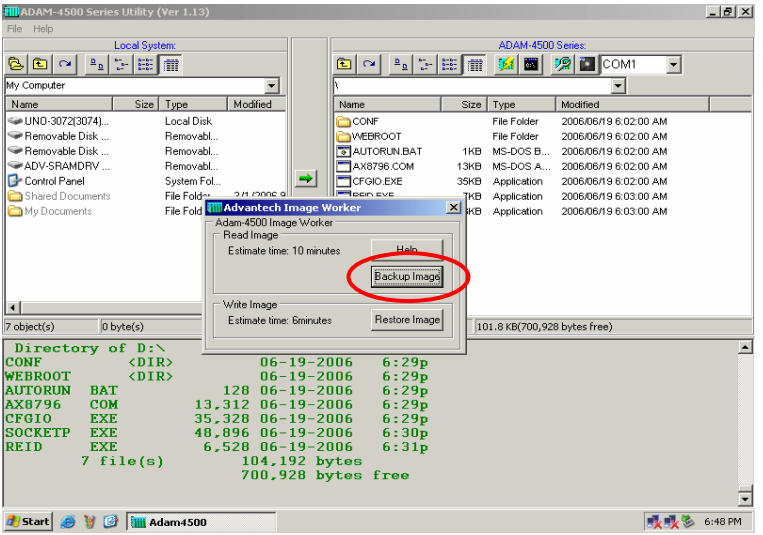
2. Click the “Refresh” button. You can see latest files in drive D of ADAM 4500 Series Controller by the right window.



3. Click the “Advantech Image Worker” button to perform the backup function.

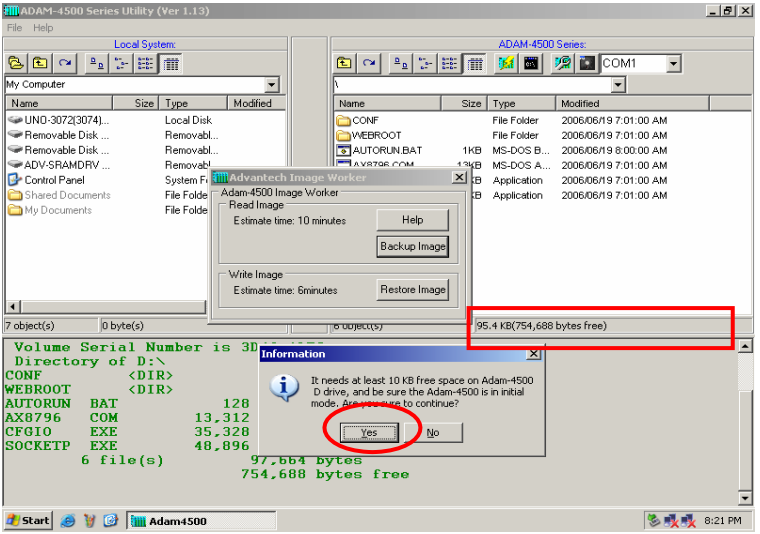


4. Click the “Backup Image” button.

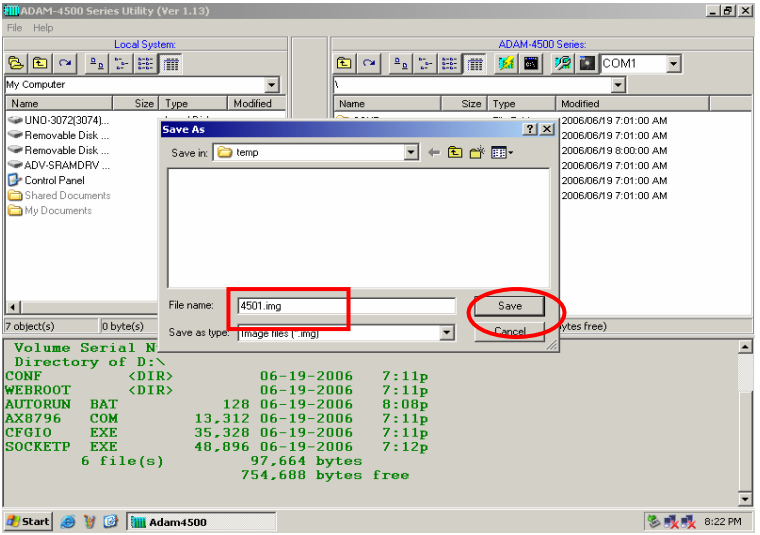


5. There will be warning dialog window showing. Check if there is

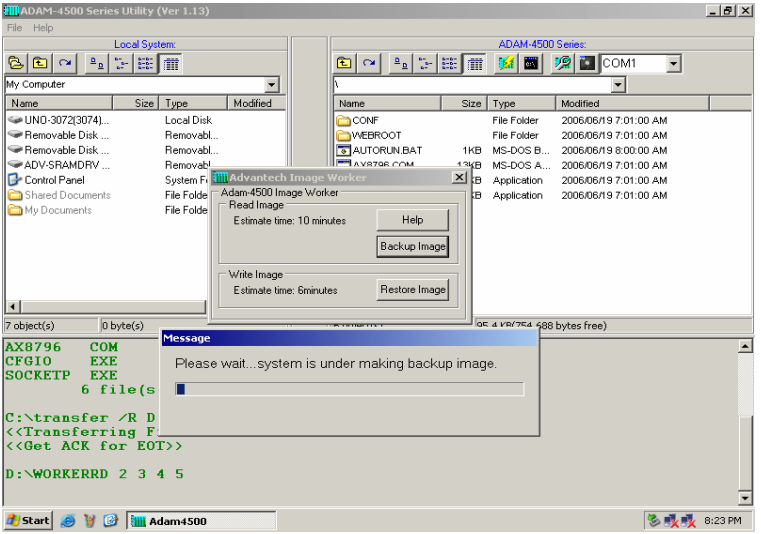
10KB free space on drive D of ADAM-4500 Series Controller by the status bar at the bottom of right window. If the size is greater than 10KB, click the “Yes” button to continue.



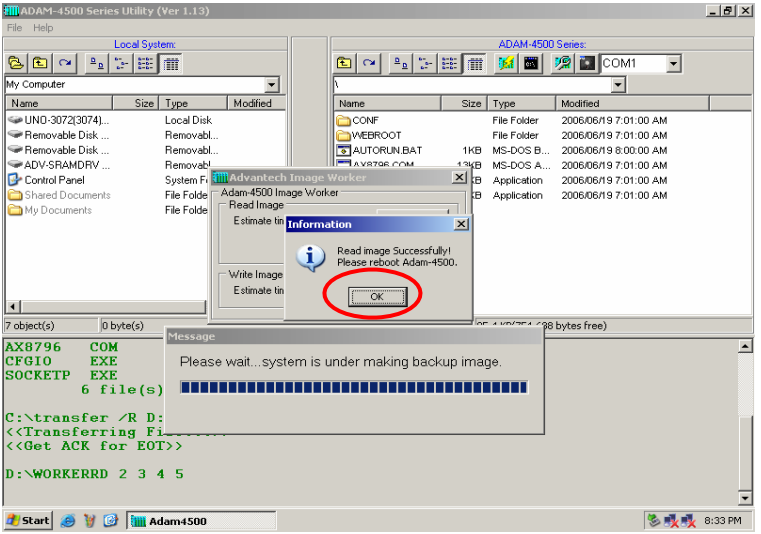
6. Select where you want to save the image file and type the image file name. Then click the “Save” button.



7. Backup function will start to process.



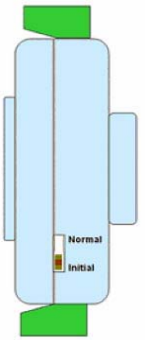
8. Click the “OK” button to finish the backup process.



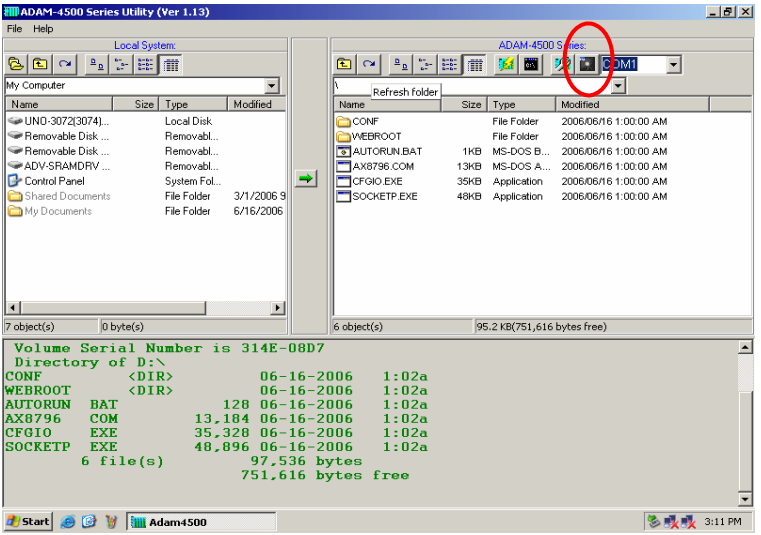
3.8 Restore the drive D from image file.

Following steps will demonstrate how to restore image file to drive D. In this example, we will use the image file produced in section 3.7.

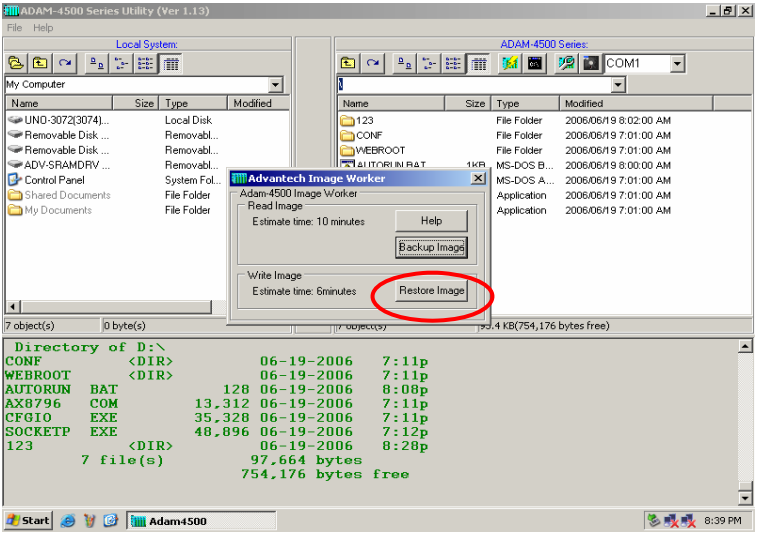
1. Put switch into **Initial mode** and then reboot.



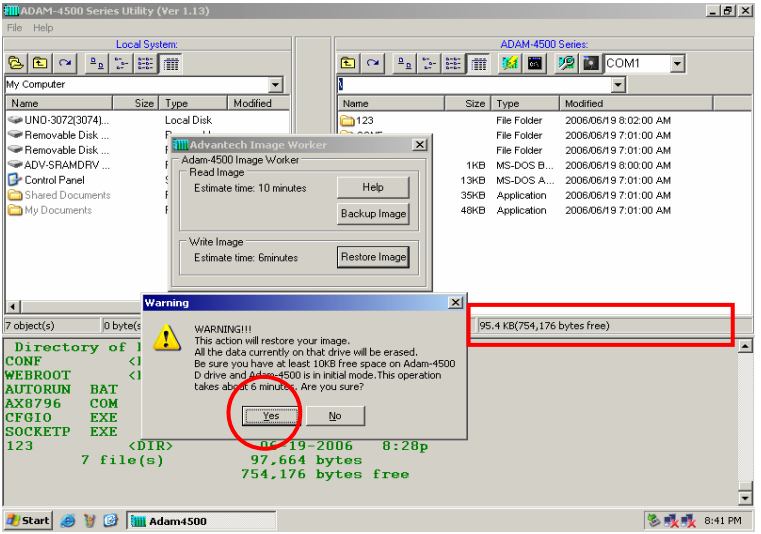
2. Click the “Advantech Image Worker” button to perform the restore function.



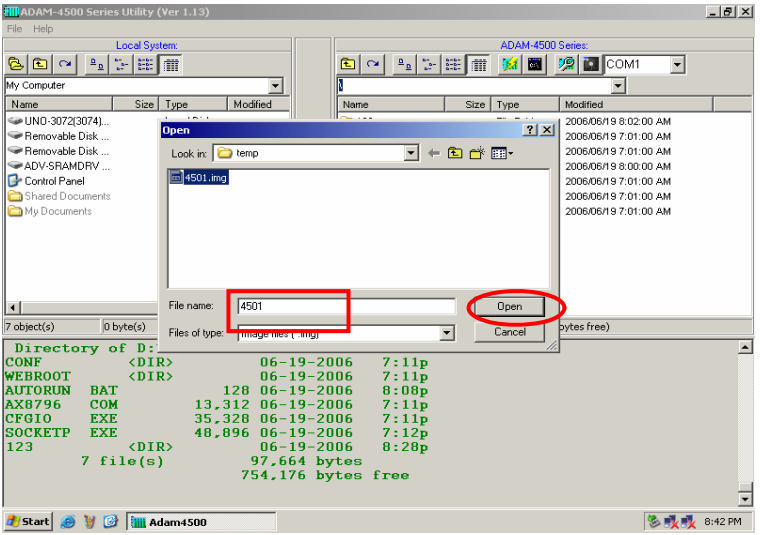
3. Click the “Restore Image” button.



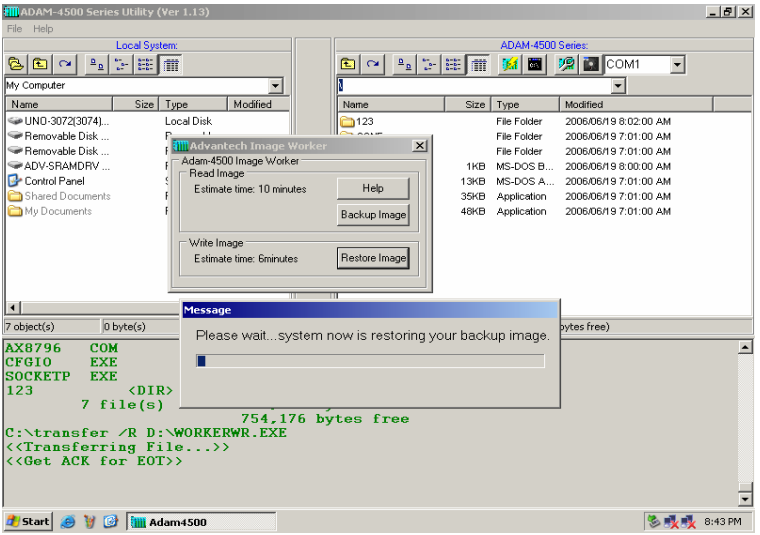
4. There will be warning dialog window showing. Make sure all files on drive D can be deleted. Also make sure there is 10KB free space on drive D of ADAM-4500 Series Controller by the status bar at the bottom of the right window. If everything is okay, click the “Yes” button.



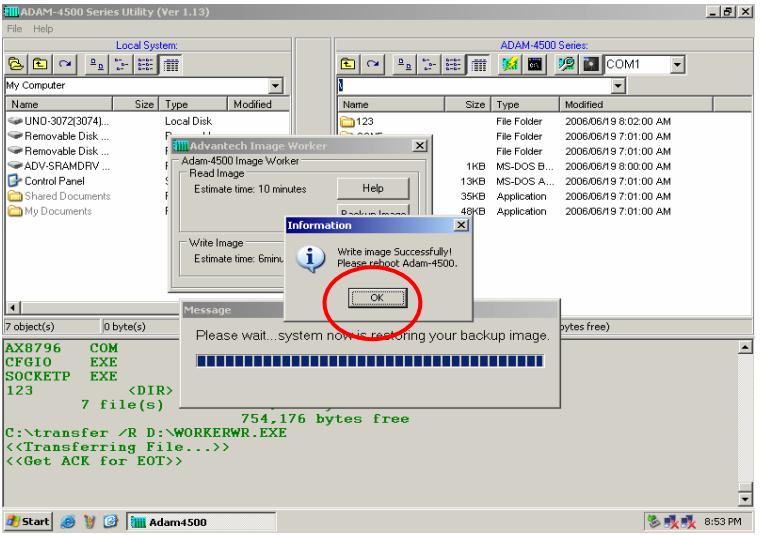
5. Select the image file you want to reload. (Here we use the image file created in 3.7) Click the “Open” button.



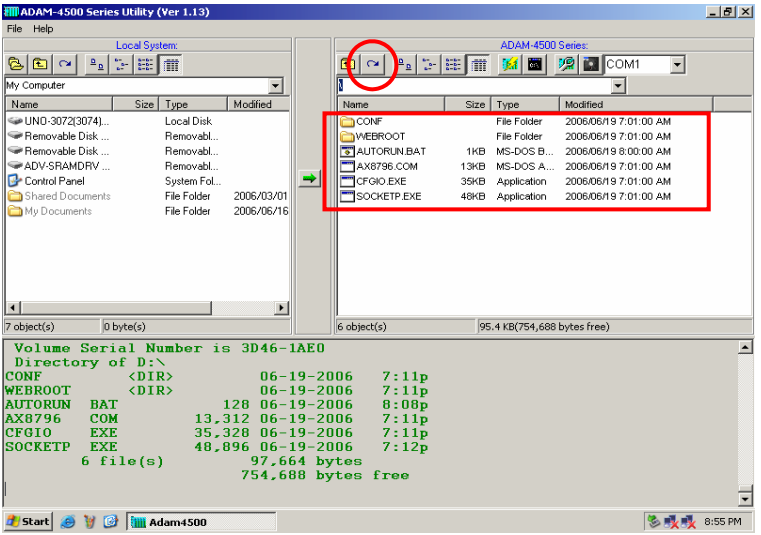
6. Restore function will start to process.



7. Click the “OK” button to finish the restore process. Reboot the ADAM-4500 Series Controller.



8. Click the “Refresh” button. Check the drive D has been restored from the backup image file.


ADAM-4500 Series Utility (Ver 1.13)
File Help
Local System:
ADAM-4500 Series:
COM1
CONF File Folder 2006/06/19 7:01:00 AM
WEBROOT File Folder 2006/06/19 7:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/06/19 8:00:00 AM
AX8796.COM 13KB MS-DOS A... 2006/06/19 7:01:00 AM
CFGIO.EXE 35KB Application 2006/06/19 7:01:00 AM
SOCKETP.EXE 48KB Application 2006/06/19 7:01:00 AM
Volume Serial Number is 3D46-1AE0
Directory of D:\
CONF <DIR> 06-19-2006 7:11p
WEBROOT <DIR> 06-19-2006 7:11p
AUTORUN BAT 128 06-19-2006 8:08p
AX8796 COM 13,312 06-19-2006 7:11p
CFGIO EXE 35,328 06-19-2006 7:11p
SOCKETP EXE 48,896 06-19-2006 7:12p
6 file(s) 97,664 bytes
754,688 bytes free
Start
Adam4500
8:55 PM

4

Guidelines for Network Functions

The network features of ADAM-4500 Series Controller are very rich. In order to shorten the learning time about versatile network features, the network functions will be present by step-by-step demonstration in this chapter. The detail information of related functions, utilities and applications are shown on later chapters. The sample programs can also be found after ADAM-4500 Series Controller utility on ADAM CD
is installed.

Before you start to test the network functions, you have to configure two files as following.

SOCKET.CFG: Text file contains related configuration command.
SOCKET.UPW: Text file contains user name and password.

SOCKETS Configuration Files: SOCKET.CFG, HOSTS

SOCKETS use two files in the D:\CFG directory (default) or any other directory specified by the SOCKETS environment variable. These files are SOCKET.CFG, the default start-up file, and HOSTS, the host names file. If not found, SOCKETS uses the default SOCKET.CFG in the D:\CFG directory.

SOCKET.CFG is a text file containing configuration commands. Empty lines and lines starting with # are ignored. Commands are used
to specify protocol parameters like the IP address of the stack, interface parameters like Packet Driver or Asyncronous Serial lines,
routes and various other parameters. Here is a simple example:

IP address demo
    Set the IP address of this host to 192.6.1.1.

Interface pdr if0 dix 1500 5
    Use Packet Driver, naming the interface ‘if0’, MTU=1500, Receive buffers = 5

Route add default if0 router
    Route all traffic to unknown destinations via ‘if0’ using ‘router’ as a gateway

TCP mss 1460
    TCP Maximum Segment Size = 1460.

TCP window 2920
TCP Maximum window = 2920.

Start prntserv
Start printer server on PRN using default port of 10.

HOSTS is an optional file containing mappings of IP addresses in dotted decimal notation to names.

Sample HOSTS file:

192.6.1.1 demo
192.6.1.2 router
192.6.1.3 server

SOCKET.CFG Samples

The following configuration file contains the minimum possible commands for a valid configuration file: just one. This is to specify that the interface should use a Packet Driver, the interrupt vector, which must be searched for. It should use DIX encapsulation, have an MTU of 1500 and have a maximum of 5 receive buffers. Since no IP address is specified, BOOTP will be used and the required operating parameters will be retrieved from a BOOTP server, which must be available on the network.

SOCKET.CFG:
interface pdr if0 dix 1500 5

The following is a more typical example specifying a static IP address, a Packet Driver interface, a default route, the TCP MSS and WINDOW.

SOCKET.CFG:
 # Sample configuration file
ip address 192.6.1.1
interface pdr if0 dix 1500 5
route add default if0 192.6.1.2
tcp mss 1460
tcp window 2920

Format of "SOCKET.UPW"

This is the same file used for the HTTP and FTP server's (*FTPD.EXE*) permissions. This file consists of lines where each line contains a user's information. A line starting with a # is considered a comment and is ignored. Each line consists of four fields:

<Username> <Password> <Working Directory> <Permissions> [# comment]

⌘ Username: The name of this user. If it is *, it will be used when the client does not specify a username.

⌘ Password: This user's password. If it is *, no password is required

⌘ Working Directory: The user will only have access to this directory and its subdirectories. If it is '/', this user has access to the whole system. HTTP_DIR can be referred to as '\'. If a relative path is specified, it is appended to HTTP_DIR.

⌘ Permissions: When a user is granted both FTP and HTTP permissions, the FTP permissions must appear **first**, otherwise they will be ignored. Permission are listed below:

FTP Rights:

d	change directories
c	create/delete directories
w	write files
r	read files

HTTP Rights:

e	get files
p	post files
g	use CGI
m	use remote console

Fields should be separated by single spaces. If any field is missing the entry is ignored. A comment may follow the last field (permissions) of the line.

Here are two example configuration files, which are used by the following demonstrations:

SOCKET.CFG:

```
# Packet driver settings
ip address 192.168.1.4
interface pdr if0 dix 1500 10 0x60

# The following will cause SOCKETS to display IP status
ip address

# The following lines set TCP parameters
ip ttl 64
tcp mss 1460
tcp window 2920
```

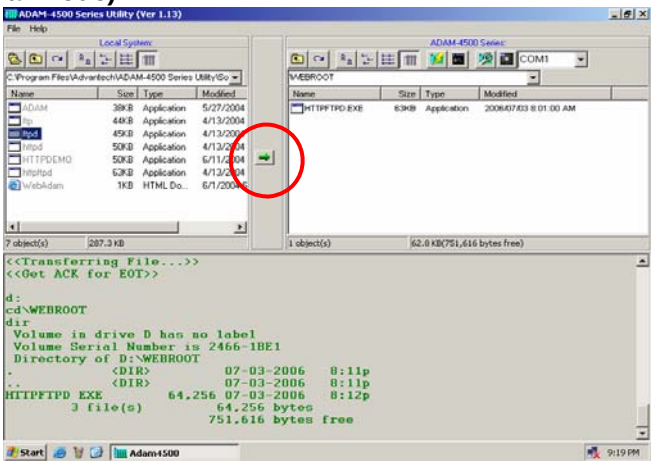
SOCKET.UPW:

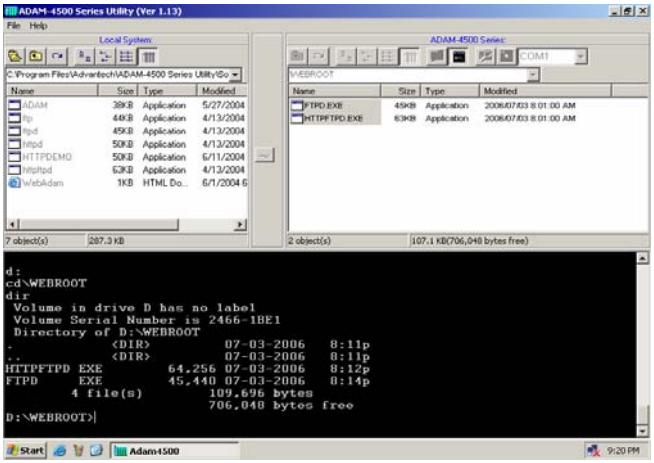
```
su su / drwcepgm     # su can do everything on whole system.

* * \guest rg              # everyone can read (FTP) and get (HTTP)
                           # from %HTTP_DIR%\guest

test1 test1 \ drep         # test1 can change directories and read files (FTP)
                           # test1 get and post files (HTTP) in %HTTP_DIR%\

ftp1 ftp1 \guest rd        # ftp1 can read files and change directories (FTP)
                           # in %HTTP_DIR%\guest
                           # ftp1 has no HTTP rights

http1 http1 / epgm         # http1 can get and post files, use CGI,
                           # and use remote console.
                           # http1 has no FTP rights

user1 user1 \user\user1 rdcw      # user1 has full FTP access rights to the
                                  # directory %HTTP_DIR%\user\user1
                                  # user1 has no HTTP rights
```

4.1 FTP Server

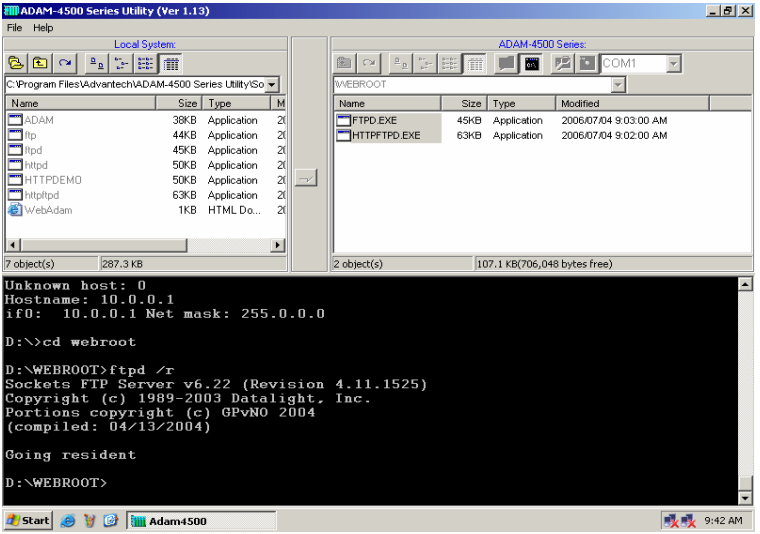
Utility: **FTPD.EXE**
System configuration:
- Running FTP server on ADAM-4500 Series Controller
- FTP Client program on host PC

1. Download FTPD.EXE (under \ADAM-4500 Series Utility\Source\ Drive_D\Extension_files on Host PC) onto drive D under "Webroot" directory. (Put switch into **Initial mode)**

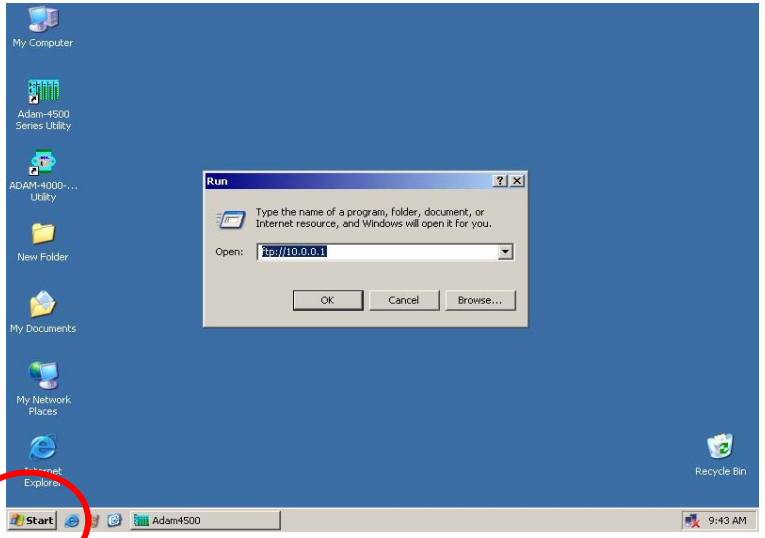




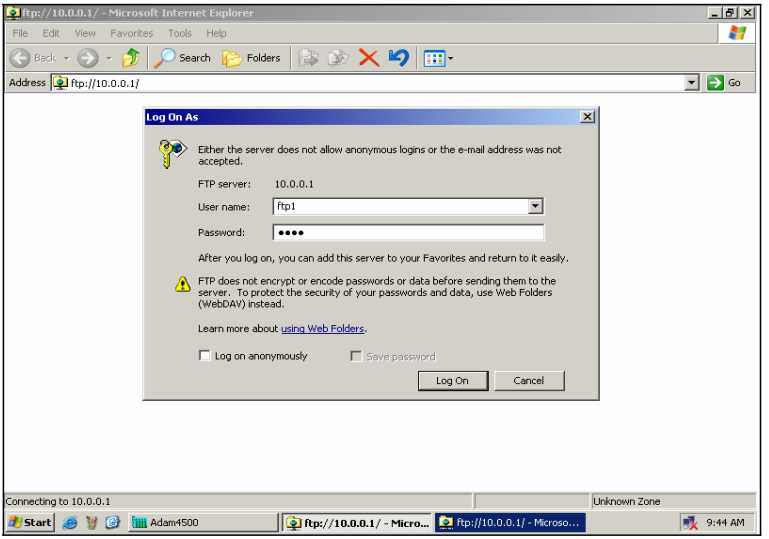
2. Put switch into **Normal mode** and then reboot. Type **"cd webroot"** to enter "Webroot" directory. Type **"ftpd /r"** to run FTPD.EXE at resident.



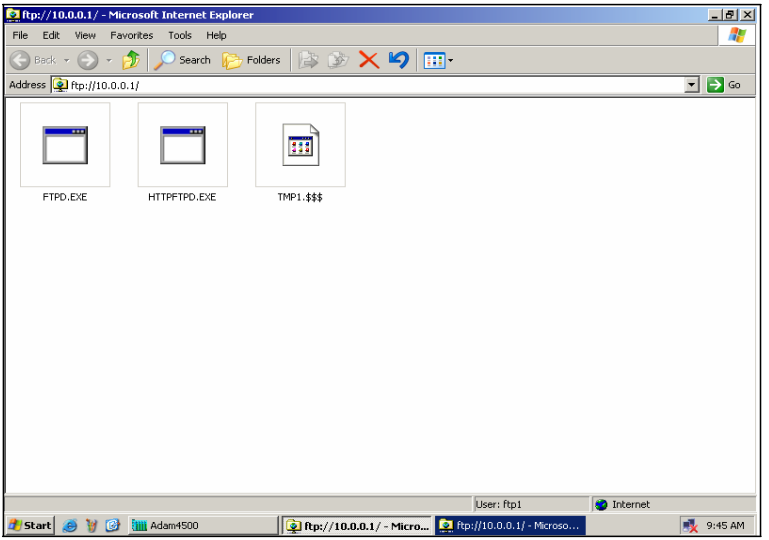
3. Connect to the FTP server on Host PC. (The IP Address of Host PC and ADAM-4500 Series Controller should be in the same domain)



4. Login FTP server by typing user name and related password
(user names and passwords are listed in **SOCKET.UPW** under
\ADAM-4500 Series Utility\Source \Drive_D\Default_files\Conf)


ftp://10.0.0.1/ - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Folders
Address ftp://10.0.0.1/
Go
Log On As
Either the server does not allow anonymous logins or the e-mail address was not accepted.
FTP server: 10.0.0.1
User name: ftp1
Password: ••••
After you log on, you can add this server to your Favorites and return to it easily.
FTP does not encrypt or encode passwords or data before sending them to the server. To protect the security of your passwords and data, use Web Folders (WebDAV) instead.
Learn more about using Web Folders.
Log on anonymously
Save password
Log On
Cancel
Connecting to 10.0.0.1
Unknown Zone
Start
Adam4500
ftp://10.0.0.1/ - Micro...
ftp://10.0.0.1/ - Microso...
9:44 AM

5. Check the files under "Webroot" directory are correctly.



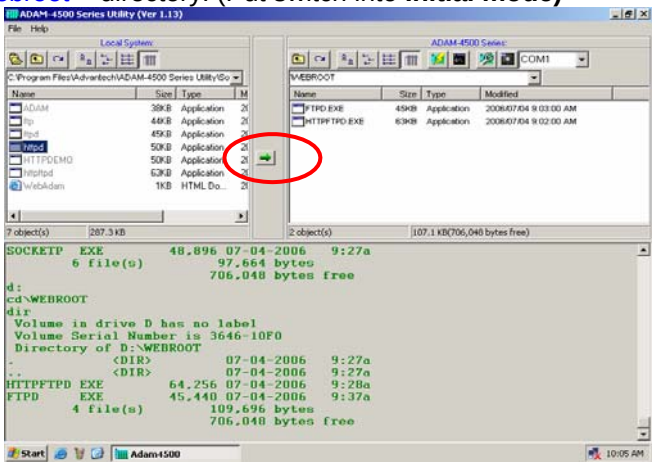
4.2 HTTP Server

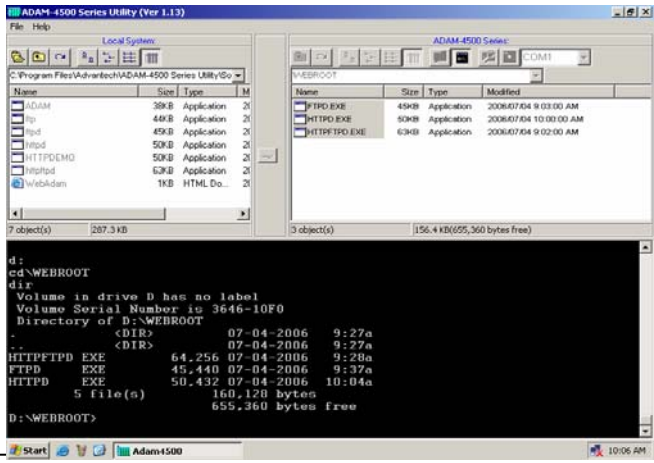
<<Example 1>>

Example program: **HTTPDEMO.EXE** (without CGI function)
Source file: **HTTPDEMO.C** under \ADAM-4500 Series Utility\Source
\Example\httpEx directory (refer to Page 4-14)
Utility: **HTTPD.EXE**
System configuration:

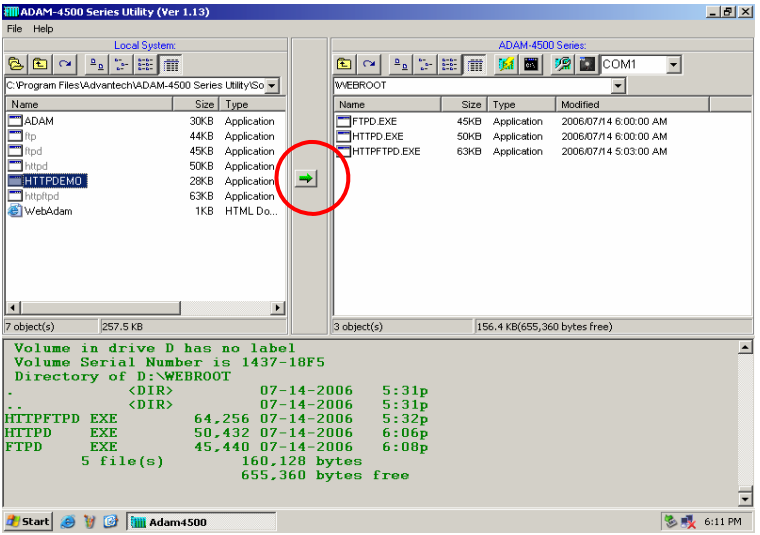
- Running HTTP server on ADAM-4500 Series Controller
- Using Web Browser to connect to the HTTP server on Host PC

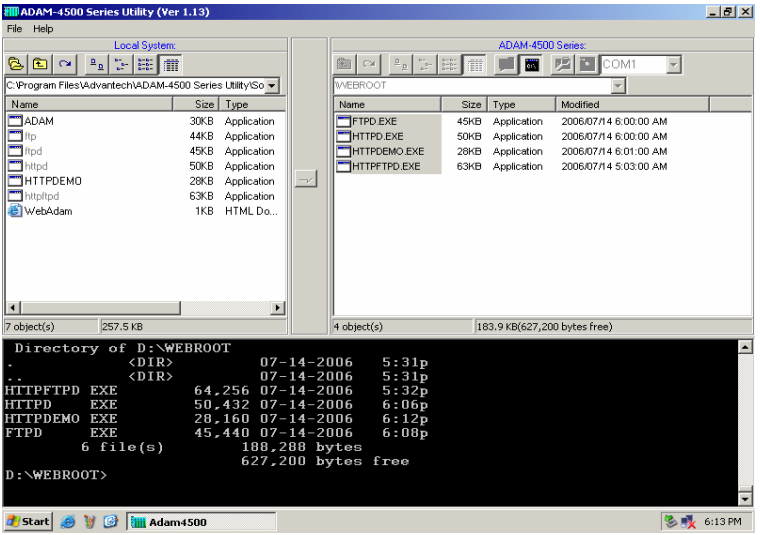
1. Download HTTPD.EXE under \ADAM-4500 Series Utility\Source\ Drive_D\Extension_files onto drive D under "Webroot" directory. (Put switch into **Initial mode)**



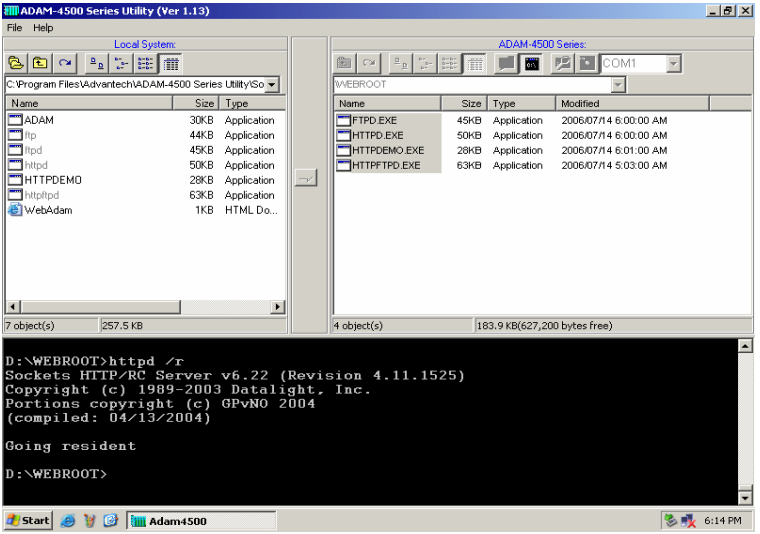


2. Using HTTPDEMO.PRJ to Build HTTPDEMO.EXE (under \ADAM-4500 Series Utility\Source\Example\httpEx) and download HTTPDEMO.EXE onto drive D under "Webroot" directory.

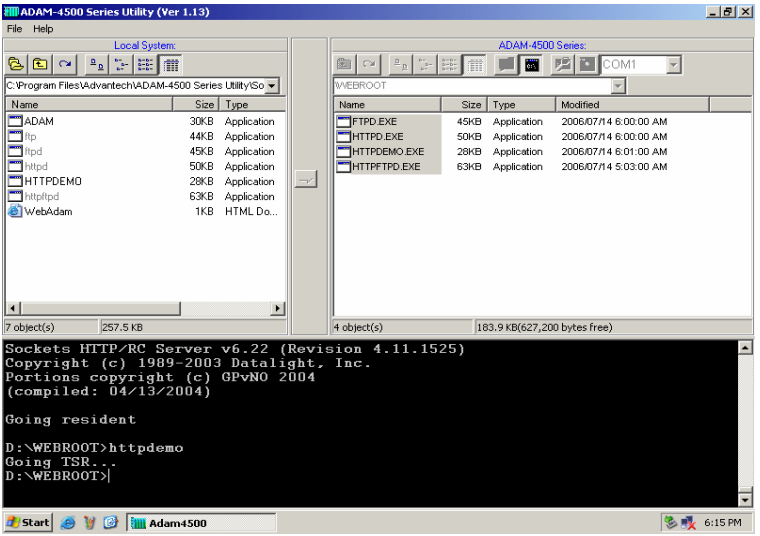




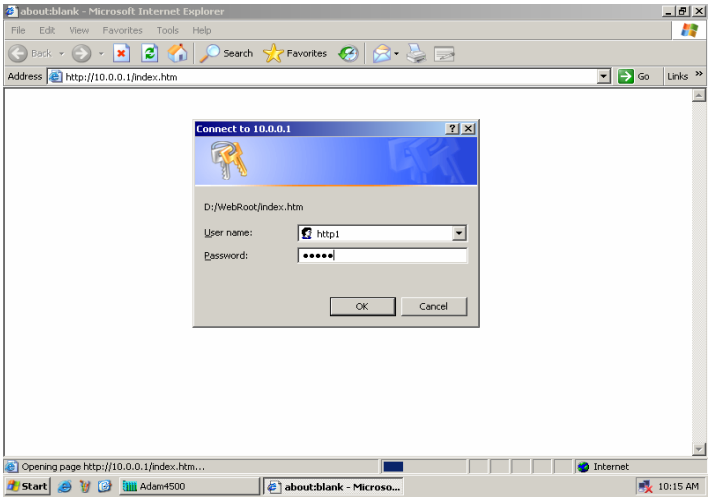
3. Put switch into **Normal mode** and then reboot. Type **"cd webroot"** to enter "Webroot" directory. Type **"httpd /r"** to run HTTPD.EXE at resident.



4. Type **"httpdemo"** to run HTTPDEMO.EXE.

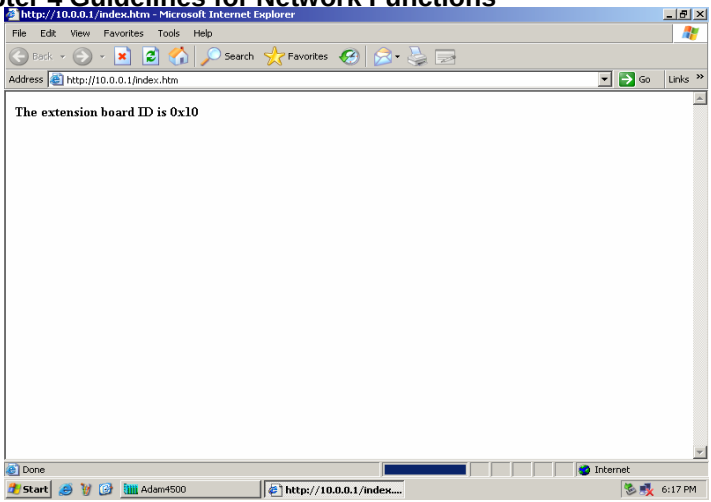


5. Run IE on Host PC, type URL **"http://10.0.0.1/index.htm"** and login in by entering user name and password.

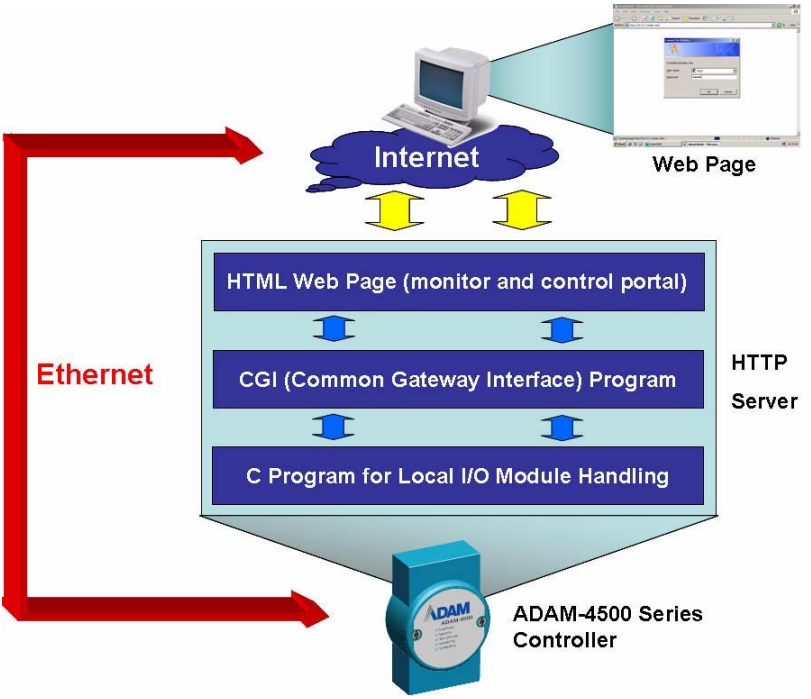


Note: 10.0.0.1 is the default IP of ADAM-4500 Series Controller, and refers to **SOCKET.CFG** under \ADAM-4500 Series Utility\ Source\Drive_D\Default_files\Conf to get real IP address. You can refer to **SOCKET.UPW** for user name and password.

6. You should be able to see ADAM-4500 Series Controller built-in I/O board ID as shown by figure below.


http://10.0.0.1/index.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://10.0.0.1/index.htm
Go Links
The extension board ID is 0x10
Done
Internet
Start
Adam4500
http://10.0.0.1/index....
6:17 PM

The following figure is the software architecture of HTTP Server function. The HTTP Server is built-in in the ADAM-4500 series Ethernet Controller. Whenever users open the IE Browser to acquire data from ADAM-4500 series controller through Internet or Intranet, it will call up the existed web pages to provide a monitor and control portal. All of the commands from the web page must be linked via a CGI program to the C control program which handle the real read/write action in I/O channels.



Basically, there are three steps in the process of Web Monitoring & Control.

1. Registration: Register a HTML name for the web page you designed
2. User login and invoke control program: After registration, users can invoke local control program by login Server
3. Local I/O activated by local control program

HTTPDEMO.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>


#include "4500drv.h"
#include "CGI_Lib.h"


FILE *fp;
int number = 0;
int count = 1;

void ReplaceStr(char *ptr_str1, char *ptr_str2, int len_str);

void main(void)
{
        char * homepage_name = "index.htm";

        if(!Http_Server_Reg(homepage_name))
                return;

   printf("Program exiting...");
   HttpDeRegister("index.htm");
}

int far Callback(HTTP_PARAMS far* psParams)
//implement your program/in this function
{

   static char *ptr_OO = 0;
   char *tmpStr = 0;
   static char Htm_Content[] = "HTTP/1.0 200 OK\r\n"
    //content of html //page, content=1
      "Content-type: text/html\r\n\r\n"       //means refreshes every 1 second
      "<html><META HTTP-EQUIV=""Refresh"" content=1>"
      "<b>The extension board ID is OOOOOOO</b><p>"
      "</html>";
```

```
    number++;
    printf("Refresh %d times...\n", number);

    if (!ptr_OO)
        ptr_OO = strstr(Htm_Content, "OOO");

    sprintf(tmpStr, "0x%X", Get_BoardID());
    ReplaceStr(ptr_OO, tmpStr, 7);

    HttpSendData(psParams->iHandle, Htm_Content, strlen(Htm_Content));
    return RET_DONE;
}

void ReplaceStr(char *ptr_str1, char *ptr_str2, int len_str)        //replace
string
{
    int i;
    for(i=0; i<len_str; i++)
        ptr_str1[i] = 32;

    for(i=0; i<strlen(ptr_str2); i++)
        ptr_str1[i] = ptr_str2[i];
}
```

<<Example 2>>

Example program: **ADAM.EXE** (with CGI function)

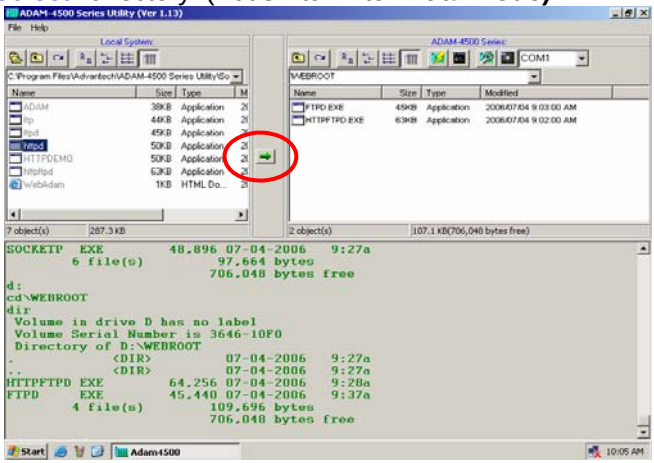
Source file: **ADAM.C** and **WEBADAM.htm** under \ADAM-4500 Series Utility\Source \Example\httpEx directory (refer to Page 4-21)

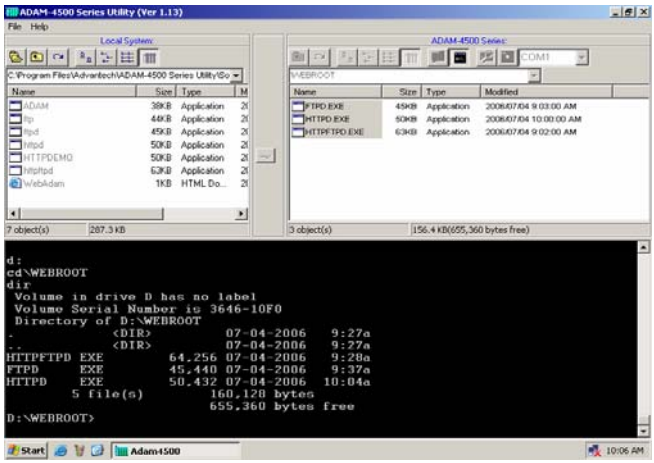
Utility: **HTTPD.EXE**

System configuration:

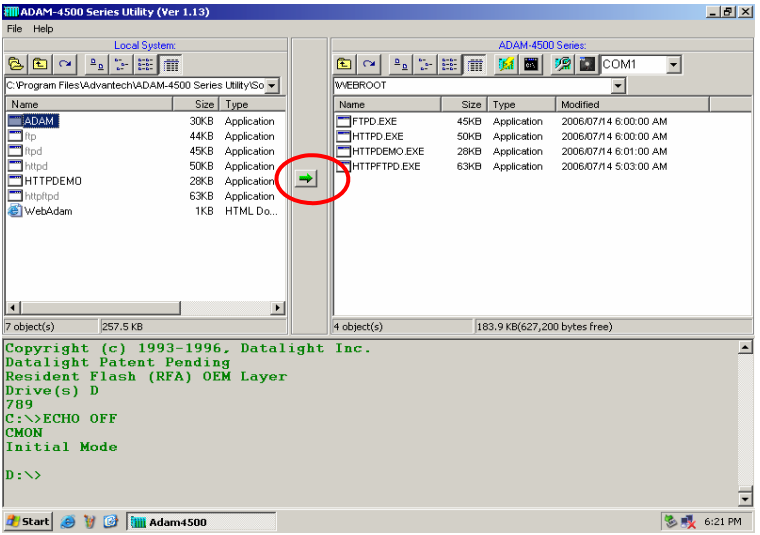
- Running HTTP server on ADAM-4500 Series Controller
- Using Web Browser to connect to the HTTP server on Host PC

1. Download HTTPD.EXE (under \ADAM-4500 Series Utility\Source\ Drive_D\Extension_files) onto drive D under "Webroot" directory. (Put switch into **Initial mode)**

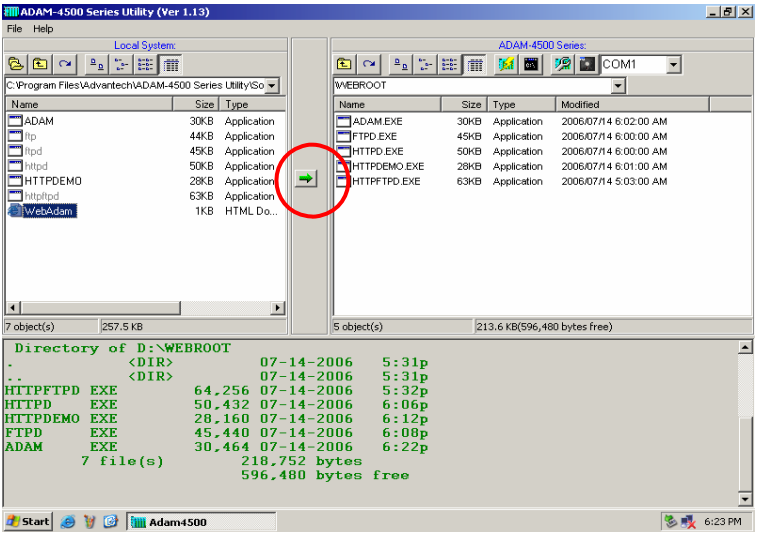




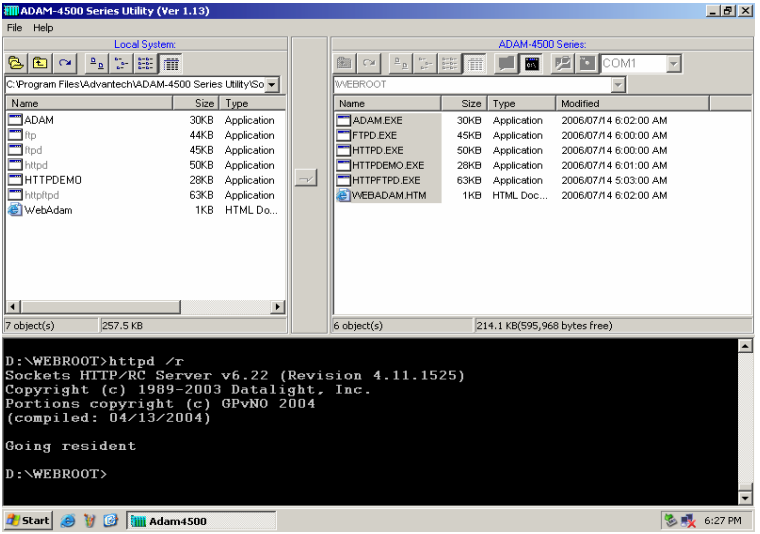
2. Using ADAM.PRJ to Build ADAM.EXE (under \ADAM-4500 Series Utility\Source\Example\httpEx) and download ADAM.EXE onto drive D under "Webroot" directory.



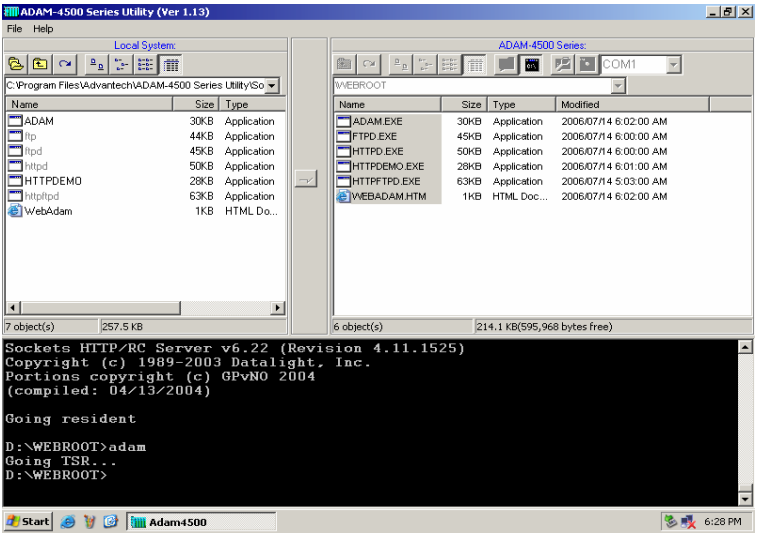
3. Download WebAdam.HTM onto drive D under "Webroot" directory.



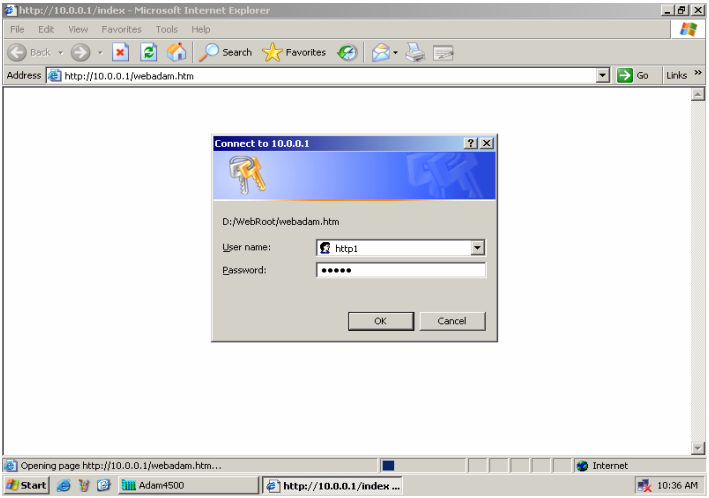
4. Put switch into **Normal mode** and then reboot. Type **"cd webroot"** to enter "Webroot" directory. Type **"httpd /r"** to run HTTPD.EXE at resident.



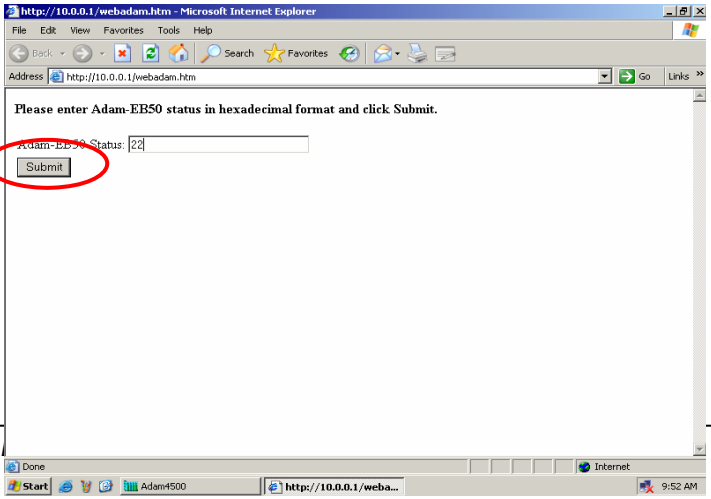
5. Type **"adam"** to run Run ADAM.EXE.



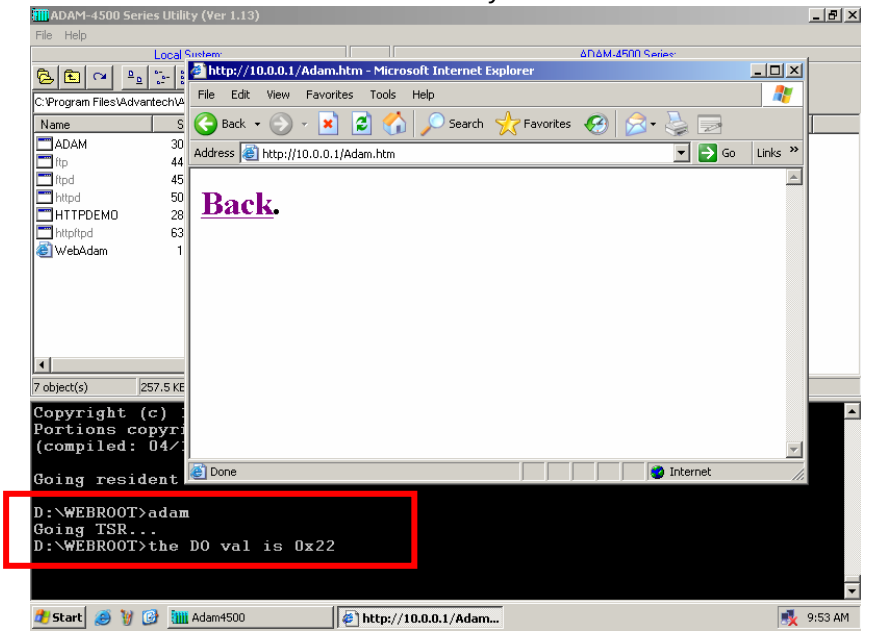
6. Run IE on Host PC, type the URL as "http://10.0.0.1/web
adam.htm" (Note: 10.0.0.1 is the default IP of ADAM-4500 Series Controller, and refer to **SOCKET.CFG** under **\ADAM-4500 Series Utility\Source\ Drive_D\Default_files\Conf** to get the real IP
address. You can refer to **SOCKET.UPW** for user name and
password)



7. You can set the status for ADAM-4500 Series Controller built-in
I/O board (EB50) by IE and then click the "Submit" button.



8. Then you can see if the EB50 status is correctly updated in
ADAM-4500 Series Controller Utility.



ADAM.C

```
#include <stdio.h>
#include <io.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>

#include "4500drv.h"
#include "CGI_Lib.h"


extern unsigned _stklen = 3000;
extern unsigned _heaplen = 2000;

int far Callback(HTTP_PARAMS far* psParams);
int returnVal(char *ptr_name, char *ptr_end);
int count = 1;


void main(void)
{
   char * homepage_name = "Adam.htm";

   if(!Http_Server_Reg(homepage_name))
      return;


   printf("Program exiting\n");
   HttpDeRegister("Adam.htm");
}

int far Callback(HTTP_PARAMS far* psParams)
//implement your program in this function
{
   char buf[200],*p,*ptr_val,*ppass;
   int iQueryLen;
   char Re_Htm_Content[400];
   char *ptr_Re = Re_Htm_Content;
   int numberbytes;
   int DoVal, DIVal;

   *buf = 0;
```

```
   iQueryLen = _fstrlen(psParams->szQuery);
   if (iQueryLen)
      _fmemcpy (buf,psParams->szQuery, iQueryLen);

  numberbytes = HttpGetData(psParams->iHandle, buf + iQueryLen, 200 -
iQueryLen);

  if (numberbytes < 0)
  {
    if (numberbytes == (-WOULDBLK))
      return RET_OK;
    else
        printf("wrong input value\n");
  }

   iQueryLen += numberbytes;


   ptr_Re += sprintf(ptr_Re, "HTTP/1.0 200 OK\r\nContent-type:
text/html\r\n\r\n<html><h1>");
   if (strncmp(buf,"DOValues=", 9) == 0) {
      ptr_val = buf + 9;
      if ((p = strchr(ptr_val,'&')) == NULL)
         printf("Please click Submit button..\n");

      printf("the DO val is 0x%x\n", returnVal(ptr_val, p));
      SetDO(EB50_ID, AllChannels, 0, returnVal(ptr_val, p));
   }
   ptr_Re += sprintf(ptr_Re, "<P><P><A
HREF=\"/WebAdam.htm\">Back</A>.</html>\n");
HttpSendData(psParams->iHandle, Re_Htm_Content, ptr_Re -
Re_Htm_Content);
   return RET_DONE;
}


int returnVal(char *ptr_name, char *ptr_end)
{
   int r_Val, buf_idx;
   char buf_val[10];

   memset(buf_val, 0, 10);
   for(buf_idx=0; buf_idx<10; buf_idx++)
```

```
    {
        if(ptr_name == ptr_end)
            break;
        buf_val[buf_idx] = ptr_name[buf_idx];
    }

    sscanf(buf_val, "%X", &r_Val);
    return r_Val;
}
```

WEBADAM.htm

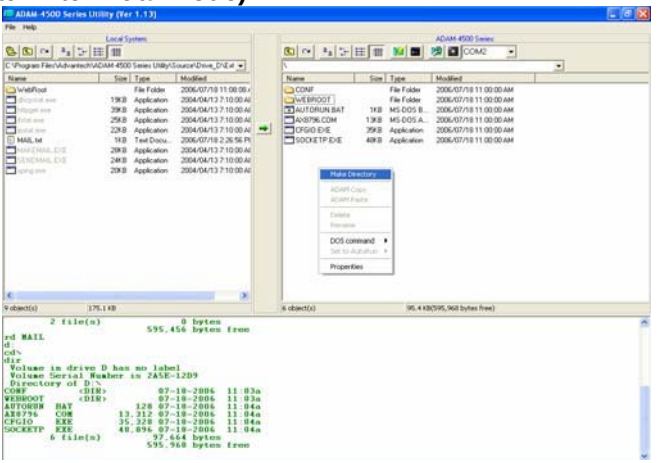
```
<html>
<head>
</head>

<body>
    <b>
    <p><p><p><p>
    Please enter I/O status in hexadecimal format and click Submit.
    </b>

    <form action="Adam.htm" method=post name="login1">

    <table>
    <tr>

    <td align=right><input name="DOValues" type=text size=30
maxlength=50></td>
    </tr>
    <tr>
    <td>
    <input name="submit" type=submit value="Submit">
    </td>
    </tr>
    </table>


</body>
</html>
```

4.3 Send Mail

Example program: **AMAIL.EXE, MAIL.TXT**
Source file: **ALARMMAIL.C** under \ADAM-4500 Series Utility\Source \Example\mail directory
Utility: **SENDMAIL.EXE, MAKEMAIL.EXE**

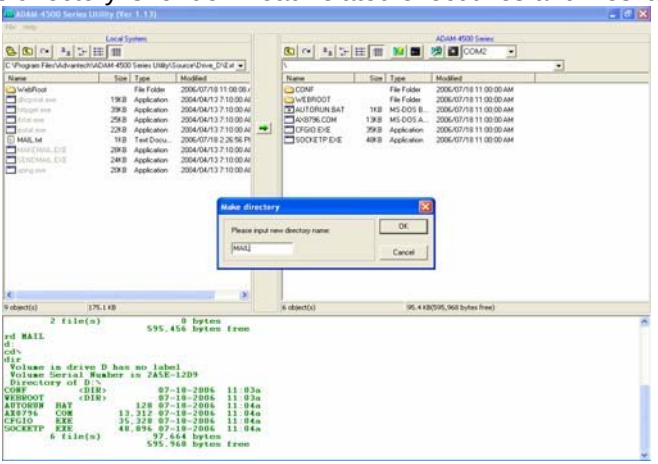
ADAM-4500 series configuration:

1. Right click on the right window and choose “Make Directory” (Put switch into **Initial mode)**

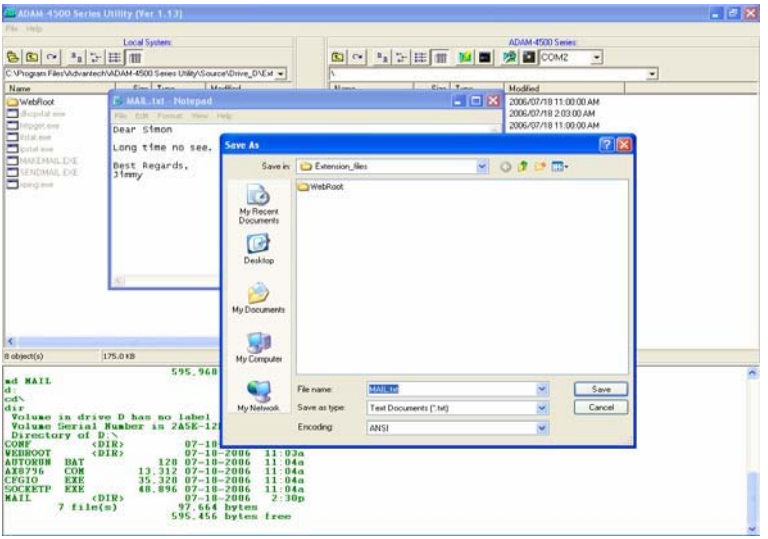


2. Type “**Mail**” to name the directory and click on the “OK” button.

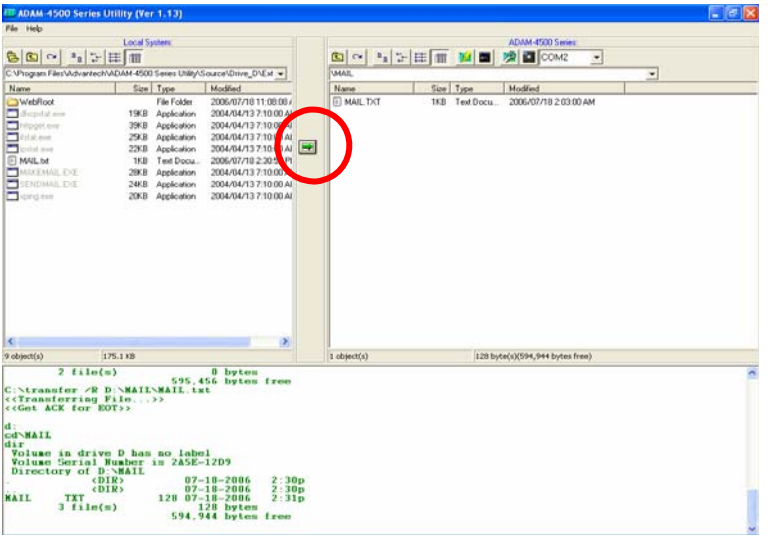
This directory is for download related executives and files later.



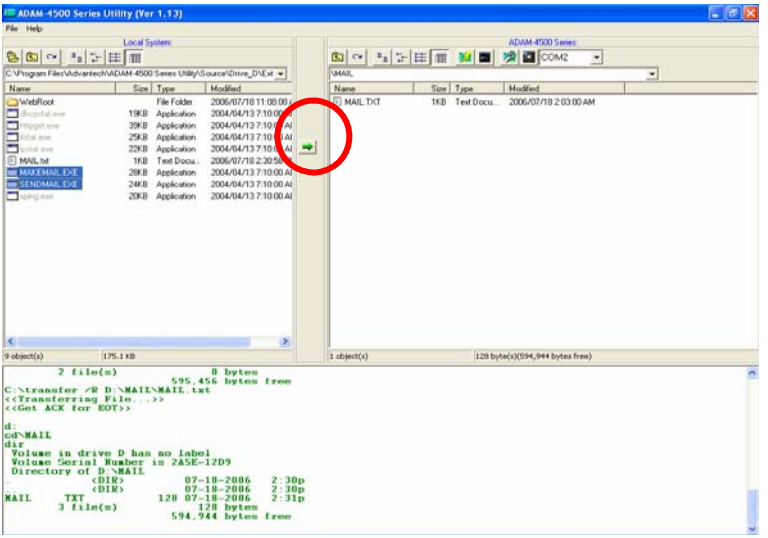
3. Create “MAIL.txt” on the left window (on the host PC). Edit the content of this file. When you finish the editing, save the file on Host PC. This file is the content file which will be sent by e-mail later.



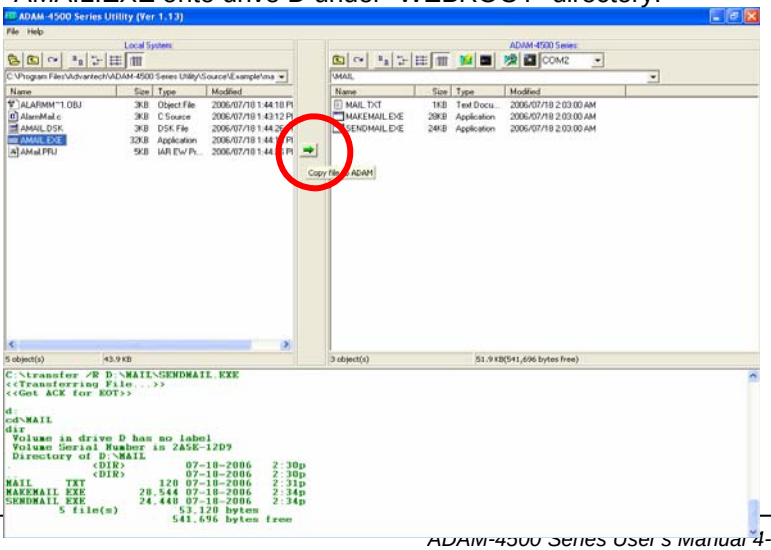
4. Download MAIL.txt onto drive D under “MAIL” directory.



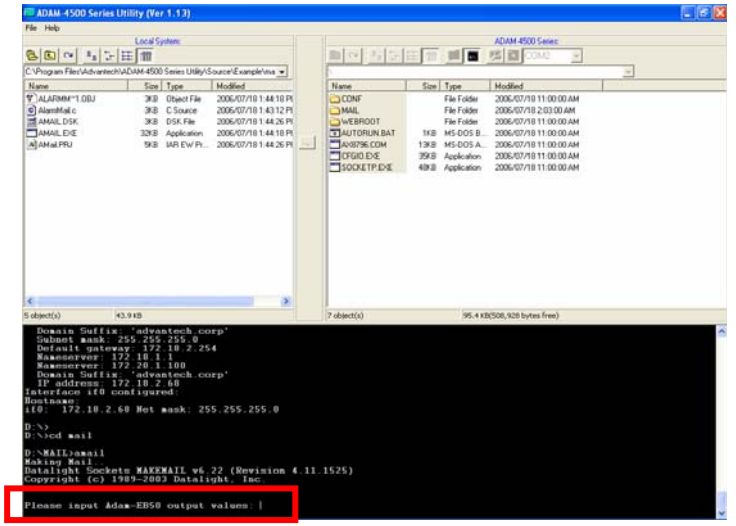
5. Download MAKEMAIL.EXE and SENDMAIL.EXE under \ADAM-4500 Series Utility\Source\Drive_D\Extension_files onto drive D under "MAIL" directory.



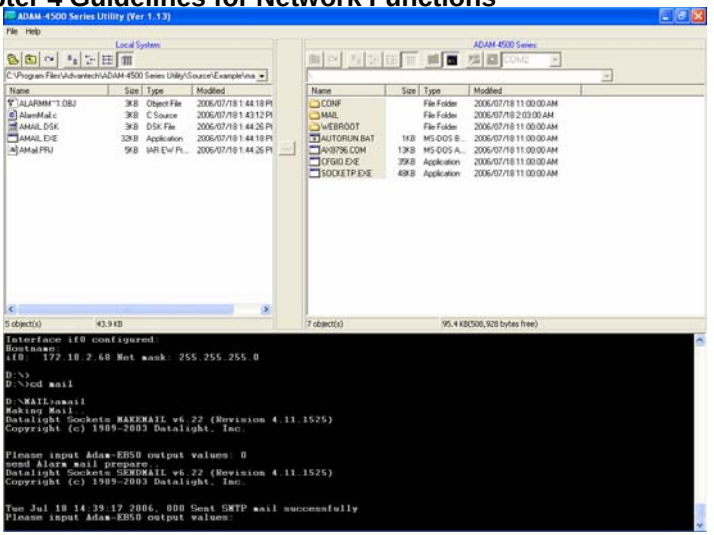
6. Using AMAIL.PRJ to Build AMAIL.EXE (under \ADAM-4500 Series Utility\Source\Example\mail) directory and download AMAIL.EXE onto drive D under "WEBROOT" directory.



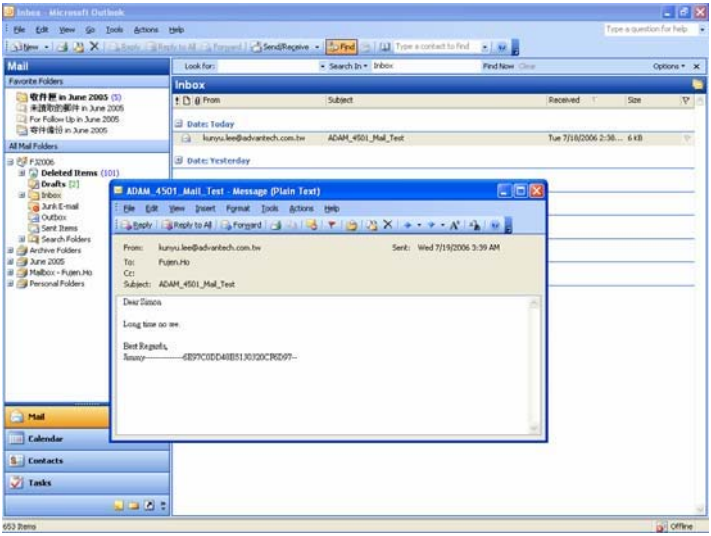
7. Put switch into **Normal mode** and then reboot. Type **"cd mail"** to enter "Mail" directory. Type **"amail"** to run AMAIL.EXE. This executive will execute MAKEMAIL.EXE to build e-mail content.
Type the DO output value to trigger the action to send e-mail.



8. The executive will execute SENDMAIL.EXE to send the e-mail content created by previous step.


ADAM-4500 Series Utility (Ver 1.13)
File Help
Local System:
ADAM-4500 Series:
Name
Size
Type
Modified
ALARMM~1.OBJ
AlamMail.c
AMAIL.DSK
AMAIL.EXE
AMail.PRJ
CONF
MAIL
WEBROOT
AUTORUN.BAT
AX8796.COM
CFGIO.EXE
SOCKETP.EXE
Interface if0 configured:
Hostname:
if0: 172.18.2.68 Net mask: 255.255.255.0
D:\>
D:\>cd mail
D:\MAIL>amail
Making Mail...
Datalight Sockets MAKEMAIL v6.22 (Revision 4.11.1525)
Copyright (c) 1989-2003 Datalight, Inc.
Please input Adam-EB50 output values: 0
send Alarm mail prepare...
Datalight Sockets SENDMAIL v6.22 (Revision 4.11.1525)
Copyright (c) 1989-2003 Datalight, Inc.
Tue Jul 18 14:39:17 2006, 000 Sent SMTP mail successfully
Please input Adam-EB50 output values:

9. Check the mailbox and it should receive the email correctly.



Note: The IP address of ADAM-4500 series should be at the same domain with the IP address of mail server, which will help you to send out the email from ADAM-4500 series. If you ask another mail server whose IP address is not at the same domain, the mail server will verify the IP address of the email sending from and then stop to provide service for sending out the email for ADAM-4500 series.

ALARMMAIL.C

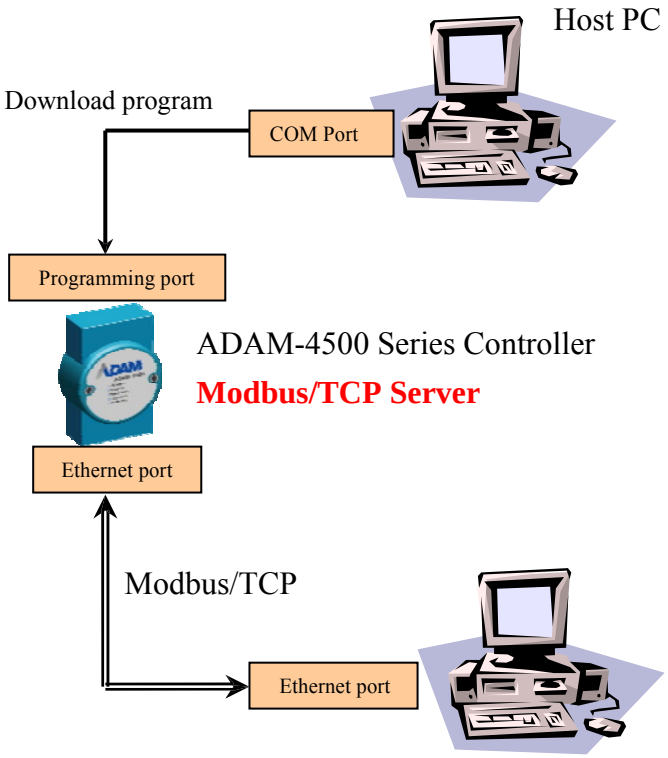
```
#include <stdio.h>
#include <process.h>
#include <errno.h>
#include "4500drv.h"

int  SendAlarmMail(void);
int MakeAlarmMail(void);
int count = 1;
void main(void)
{
   unsigned long div, dov;
        unsigned int tmpcnt;

        if(!MakeAlarmMail())
        {
                printf("make mail fail..");
                return;
        }

        while(1)
        {
                printf("Please input digital output values: ");
                scanf("%X", &dov);
                if(dov == 0x33)
                        return;
           SetDO(EB50_ID, AllChannels, 0, dov);
                printf("DO value 0x%X, press any key to continue\n", dov);
getch();
           GetDIO(EB50_ID, AllChannels, 0, &div);

                if(div == 0x000f)
                {
                        if(!SendAlarmMail())
                        {
                                printf("send mail error..");
                                return;
                        }
                }
        }

}
```

```
int MakeAlarmMail(void)
{
        char * arg_To = "-t567@123.com";
        char * arg_From = "-f345@hotmail.com";
        char * arg_subject = "-s5510TCP";
        char * arg_MailContent = "-bmail.txt";
        char * arg_O_mail = "-omail.dat";

        printf("Making Mail..\n");
        if(spawnlp(P_WAIT,
                        "d:\\mail\\makemail.exe",
                        "d:\\mail\\makemail.exe",
                        arg_To,
                        arg_From,
                        arg_subject,
                        arg_MailContent,
                        arg_O_mail,
                        NULL)==-1)
        {
                return 0;
        }

        return 1;

}
int SendAlarmMail(void)
{
        char * arg1 = "smtp.123.com";
        char * arg2 = "mail.dat";

        printf("send Alarm mail prepare..\n");
        if(spawnlp(P_WAIT,"d:\\mail\\sendmail.exe","d:\\mail\\sendmail.ex
e",arg1,arg2,NULL)==-1)
        {
                return 0;
        }

        return 1;
}
```

Note: Please refer to Chapter 6 Page 6-10 and 6-11 to see more detail about Mail Function.

4.4 Modbus/TCP Server

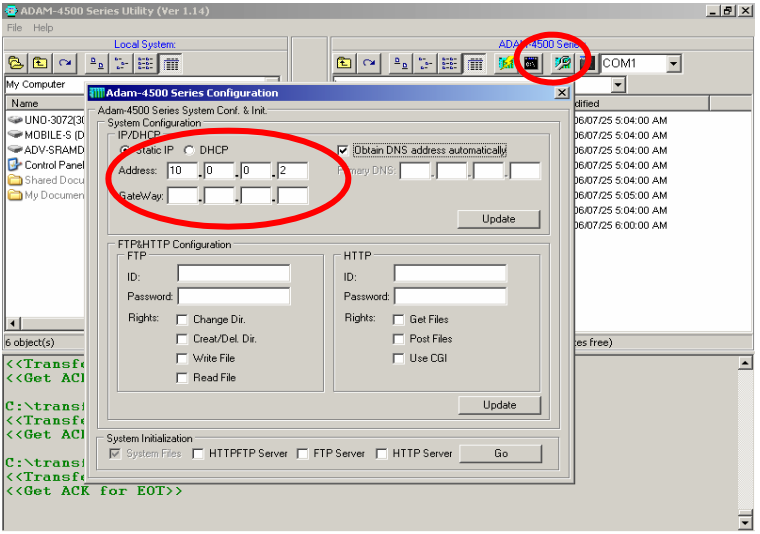
Example program: **DEMOTS.EXE**
Source file: **DEMOTS.C** under \ADAM-4500 Series Utility\Source \Example\DemoModbus directory
Hardware configuration: see figure below



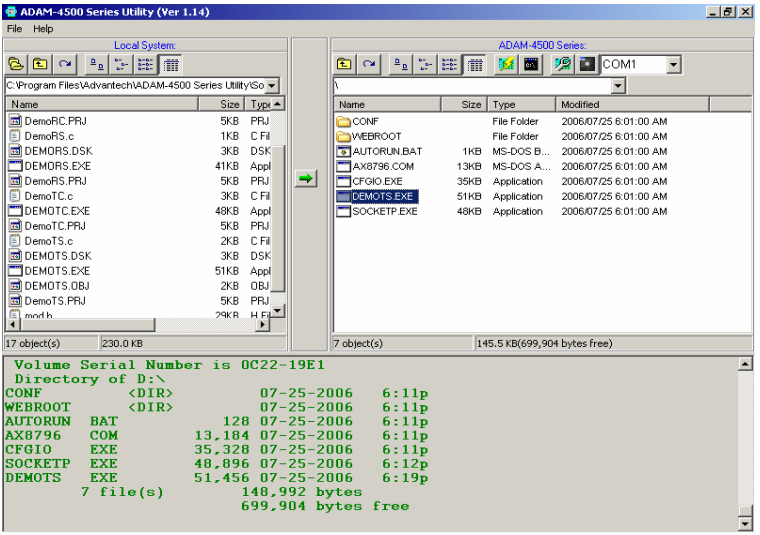
This example uses ADAM-4500 Series Controller as Modbus/TCP server, and a PC on the same Ethernet network is Modbus/TCP client. ADAM-4500 Series Controller writes on-board DIO value to address 40001, and Modbus/TCP client application such as OPC Server or ADAMVIEW read the value from that address. (Address 0 of share memory represents the address 40001 of

Modbus address)

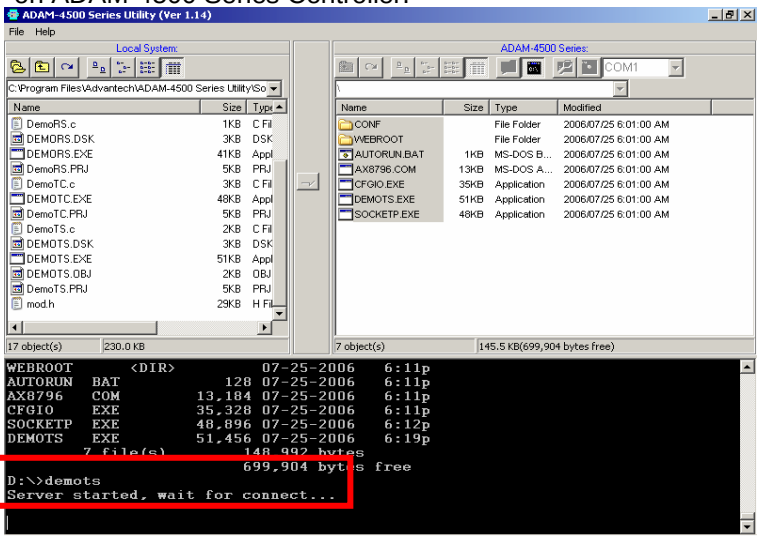
1. Click the “Adam-4500 Configuration” button to set IP address of ADAM-4500 Series Controller (Put switch into **Initial mode)**



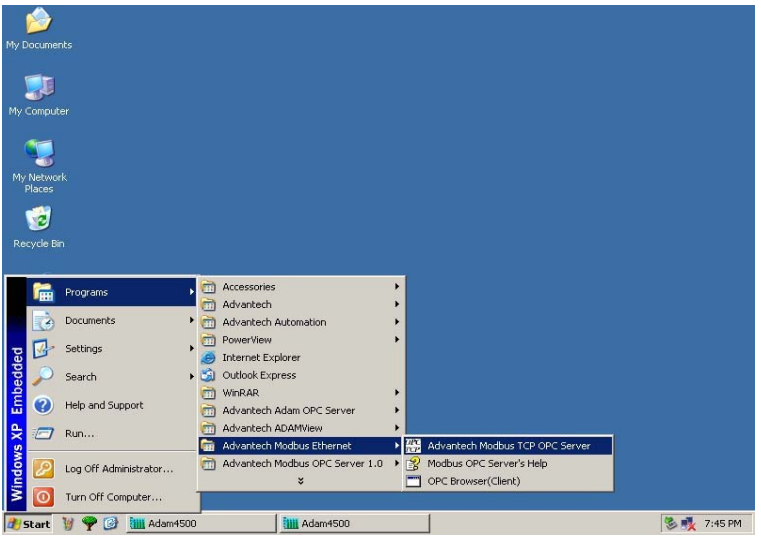
2. Build DEMOTS.EXE from DEMOTS.PRJ under \ADAM-4500 Series Utility\Source\Example\DemoModbus directory and download DEMOTS.EXE onto drive D under root directory.



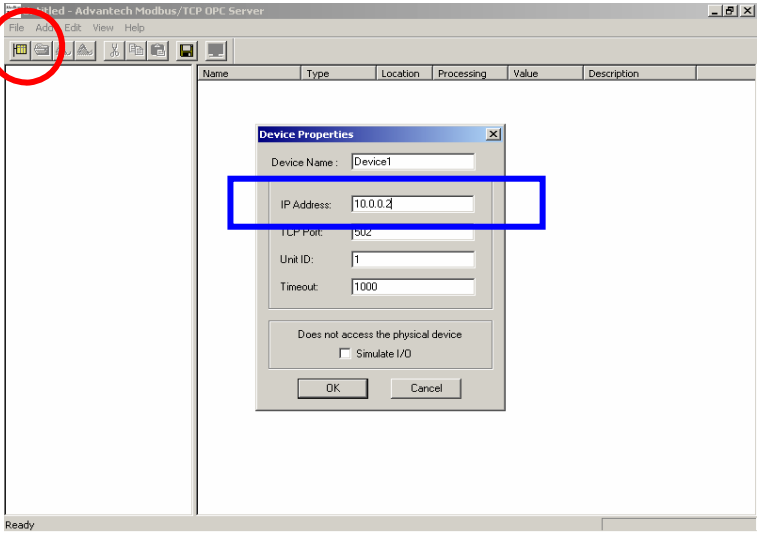
3. Put switch into **Normal mode** and then reboot. Under the root directory, run DEMOTS.EXE. This will launch Modbus/TCP server on ADAM-4500 Series Controller.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
ADAM-4500 Series:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type
DemoRS.c 1KB C Fil
DEMORS.DSK 3KB DSK
DEMORS.EXE 41KB Appl
DemoRS.PRJ 5KB PRJ
DemoTC.c 3KB C Fil
DEMOTC.EXE 48KB Appl
DemoTC.PRJ 5KB PRJ
DemoTS.c 2KB C Fil
DEMOTS.DSK 3KB DSK
DEMOTS.EXE 51KB Appl
DEMOTS.OBJ 2KB OBJ
DemoTS.PRJ 5KB PRJ
mod.h 29KB H Fil
17 object(s) 230.0 KB
COM1
Name Size Type Modified
CONF File Folder 2006/07/25 6:01:00 AM
WEBROOT File Folder 2006/07/25 6:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/07/25 6:01:00 AM
AX8796.COM 13KB MS-DOS A... 2006/07/25 6:01:00 AM
CFGIO.EXE 35KB Application 2006/07/25 6:01:00 AM
DEMOTS.EXE 51KB Application 2006/07/25 6:01:00 AM
SOCKETP.EXE 48KB Application 2006/07/25 6:01:00 AM
7 object(s) 145.5 KB(699,904 bytes free)
WEBROOT <DIR> 07-25-2006 6:11p
AUTORUN BAT 128 07-25-2006 6:11p
AX8796 COM 13,184 07-25-2006 6:11p
CFGIO EXE 35,328 07-25-2006 6:11p
SOCKETP EXE 48,896 07-25-2006 6:12p
DEMOTS EXE 51,456 07-25-2006 6:19p
7 file(s) 148,992 bytes
699,904 bytes free
D:\>demots
Server started, wait for connect...

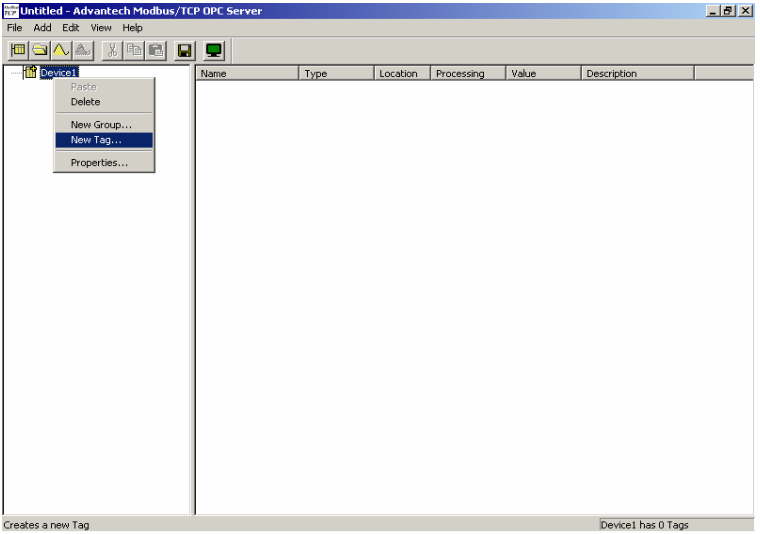
4. Launch Advantech Modbus TCP OPC Server.



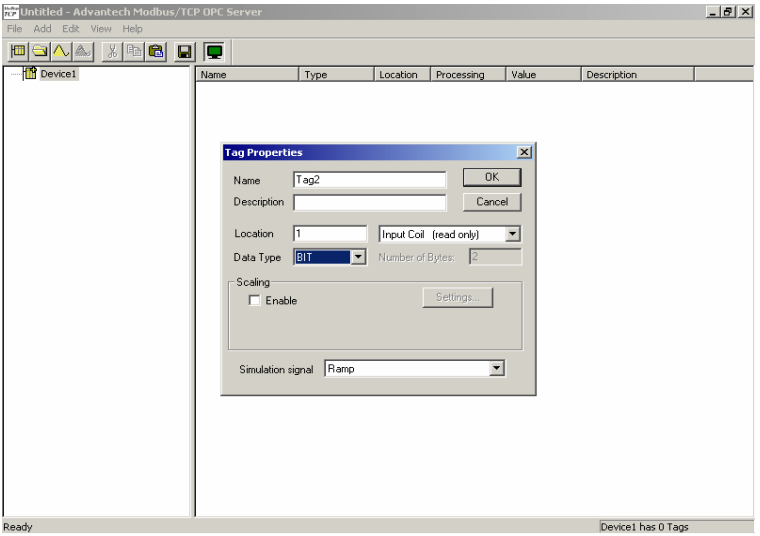
5. Create New Device under Advantech Modbus TCP OPC Server. Set the correct IP address according to step 1.



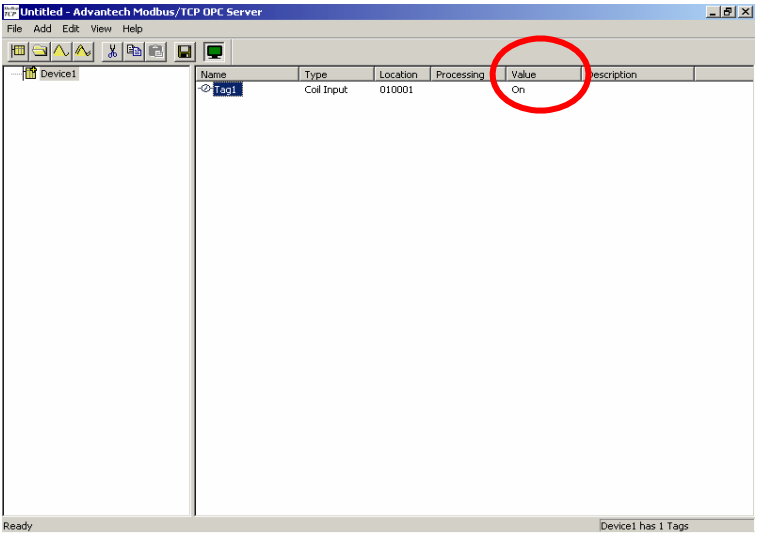
6. Right click on the new device you have just created and choose “New Tag”.



7. Configure the Tag Property.



8. You can see the value of address 40001 from Modbus/TCP server on ADAM-4500 Series Controller.


Untitled - Advantech Modbus/TCP OPC Server
File Add Edit View Help
Device1
Name Type Location Processing Value Description
Tag1 Coil Input 010001 On
Ready
Device1 has 1 Tags

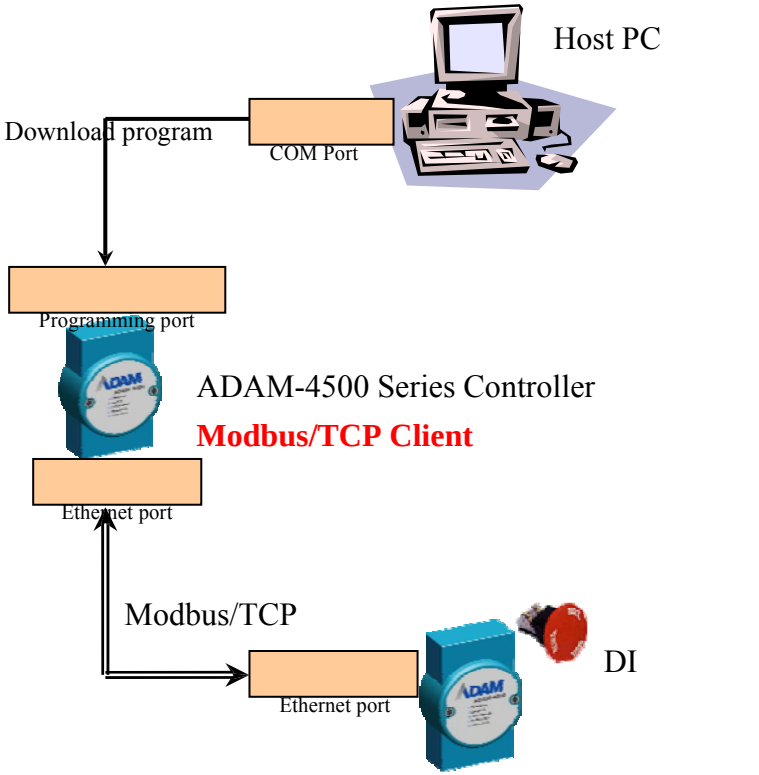
DEMOTS.C

```
#include "mod.h"
#include "4500drv.h"

#define DATASIZE 250
#define sizeofShareMem        4000

int count=0;

int main(void)
{
        SOCKET Sock_4500;
        int err_code;
        unsigned int Share_Mem[sizeofShareMem];
        unsigned int tmpcnt=0;
        int tmpidx;
        unsigned long div;


        memset(Share_Mem, 0, sizeof(Share_Mem));
        if((err_code=ADAMTCP_ModServer_Create(502, 5000, 7,

                (unsigned char *)Share_Mem, sizeof(Share_Mem)))!=0)
        //first step
        {
                printf("error code is %d\n", err_code);
        }

        Timer_Init();
        tmpidx = Timer_Set(1000);
        printf("Server started, wait for connect...\n");
        while(1)
        {
                ADAMTCP_ModServer_Update();   //second step: return 0
         //NO packet, return 1 has packet

                if(tmArriveCnt[tmpidx])
                {
                        Timer_Reset(tmpidx);
                        disable();
                        GetDIO(EB50_ID, AllChannels, 0, &div);
```

```
                    Share_Mem[0] = div;
                    //write DIO status to modbus address 40001
                    enable();
                }
        }

        ADAMTCP_ModServer_Release();

        return 0;
}
```

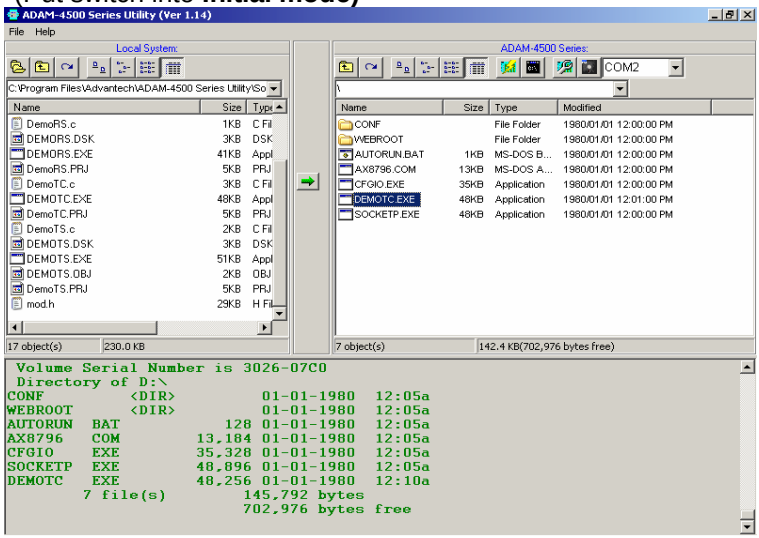
4.5 Modbus/TCP Client

Example program: **DEMOTC.EXE**
Source file: **DEMOTC.C** under \ADAM-4500 Series Utility\Source \Example\DemoModbus directory
Hardware configuration: see figure below

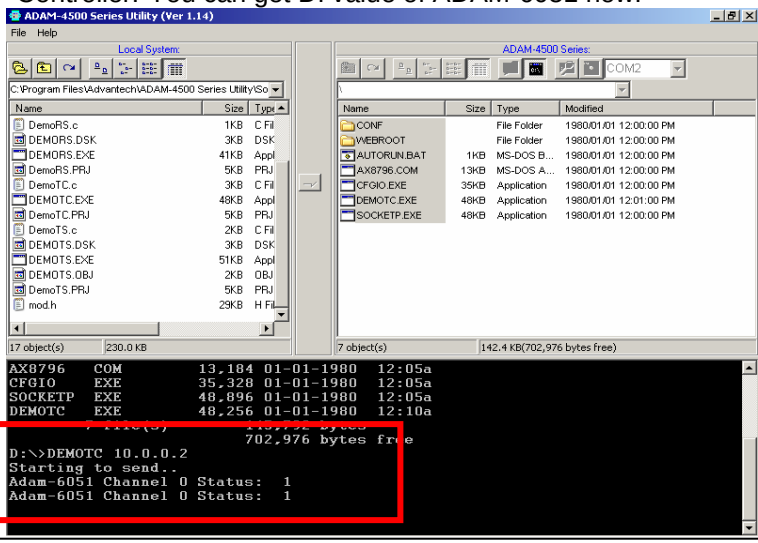


This example uses ADAM-4500 Series Controller as Modbus/TCP client to connect to the ADAM-6051 (support Modbus/TCP protocol and can be a Modbus/TCP server here), and read the DI value of ADAM-6051 through Modbus/TCP communication.

1. Build DEMOTC.EXE from DEMOTC.PRJ under \ADAM-4500 Series Utility\Source\Example\DemoModbus directory and download DEMOTC.EXE onto drive D under root directory. (Put switch into **Initial mode)**



2. Put switch into **Normal mode** and then reboot. Run DEMOTC.EXE (remember to add IP address of ADAM-6051) to launch Modbus/TPC client on ADAM-4500 Series Controller. You can get DI value of ADAM-6051 now.



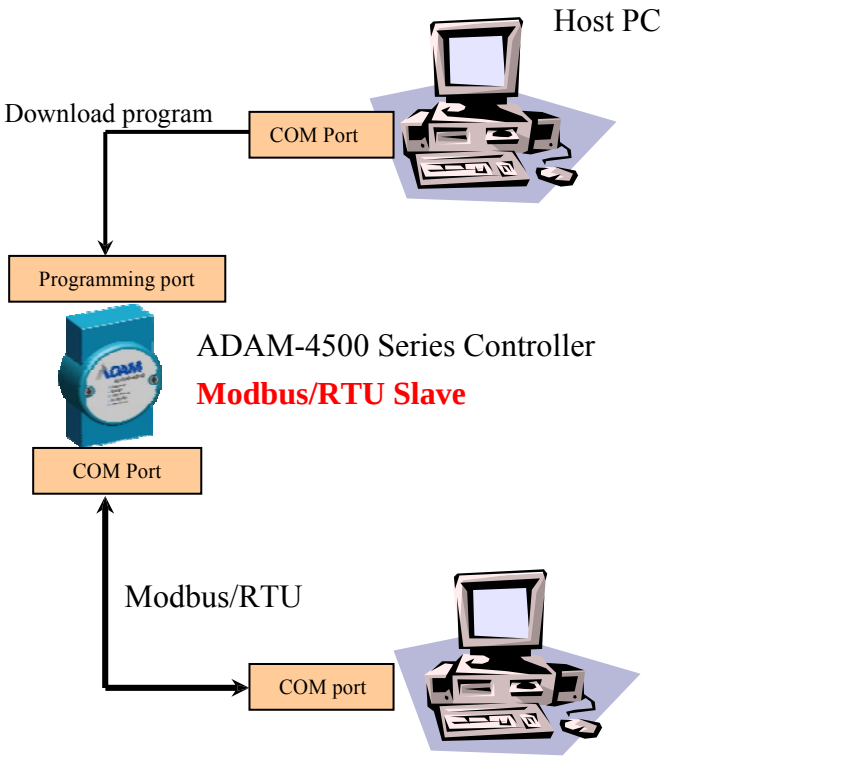
DEMOTC.C

```
#include "mod.h"

#define Server_Port 502
#define MAXDATASIZE 100


int main(int argc, char *argv[])
{
        char * ServerIP;
   SOCKET SO_4500;
   unsigned char HostData[MAXDATASIZE];
   int DataByteCount = 0;
   int tmp;
   unsigned int tmpcnt=0, tmpcnt1=0;
   int errcode;

   memset(HostData, MAXDATASIZE, 0);

   if(argc==2)
   {
        ServerIP = argv[1];
   }
   else
   {
        printf("Please input Server IP.\n");
        return 0;
   }

   if(ADAMTCP_Connect(&SO_4500, ServerIP, Server_Port)<=0)
   {
      perror("ADAMTCP_Connect()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
   }

        printf("Starting to send..\n");
        while(1)
        {
                //Query Adam-6051 Server
        if((errcode=ADAMTCP_ReadCoilStatus(&SO_4500, 50, 0x01,
0x01, 0x01, &DataByteCount, HostData))<=0)
```

```
        {
                if(errcode==TCPTimeOut_Err)
                        perror("Time Out.\n");
                else
                printf("Error: Error Code is %d\n", errcode);
           ADAMTCP_Disconnect(&SO_4500);
           return 0;
        }
        else
        {
                printf("Adam-6051 Channel 0 Status: ");
                for(tmp=0; tmp<DataByteCount; tmp++)
                {
                   printf("%2X", HostData[tmp]&0x01);
                        }
                printf("\n");
                }

                for(tmpcnt=0; tmpcnt<50000; tmpcnt++)    //delay
                {for(tmpcnt1=0; tmpcnt1<4; tmpcnt1++){}}

        }


   return 1;
}
```

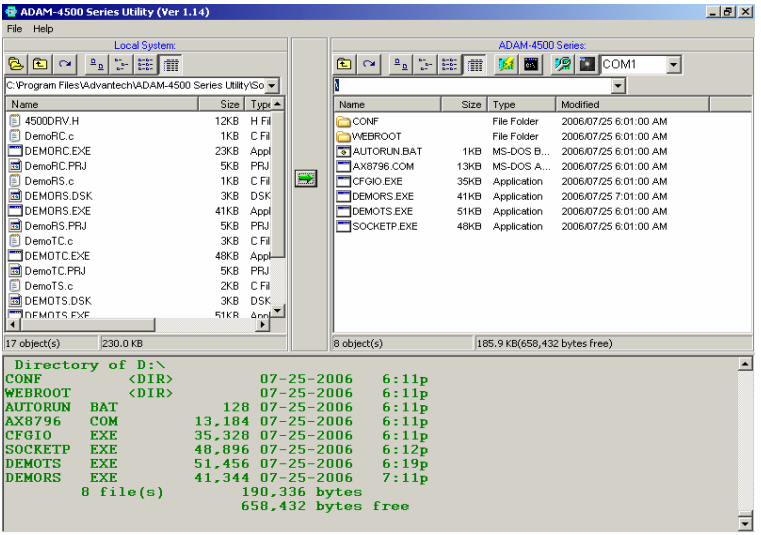
4.6 Modbus/RTU Slave

Example program: **DEMORS.EXE**
Source file: **DEMORS.C** under \ADAM-4500 Series Utility\Source \Example\DemoModbus directory
Hardware configuration: see figure below

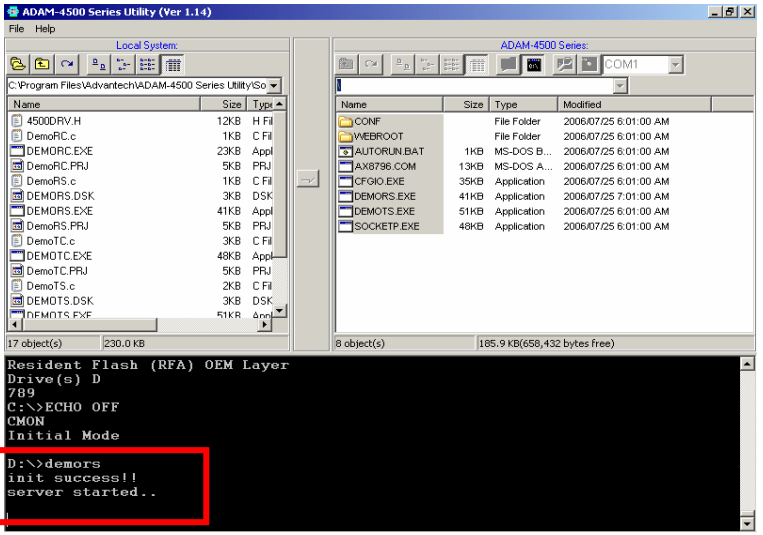


This example uses ADAM-4500 Series Controller as Modbus/RTU slave, and a PC on the same serial network is Modbus/RTU master. ADAM-4500 Series Controller writes onboard DIO value to address
40001, and Modbus/RTU client application such as OPC Server or ADAMVIEW read the value from that address. (Address 0 of share memory represents the address 40001 of Modbus address)

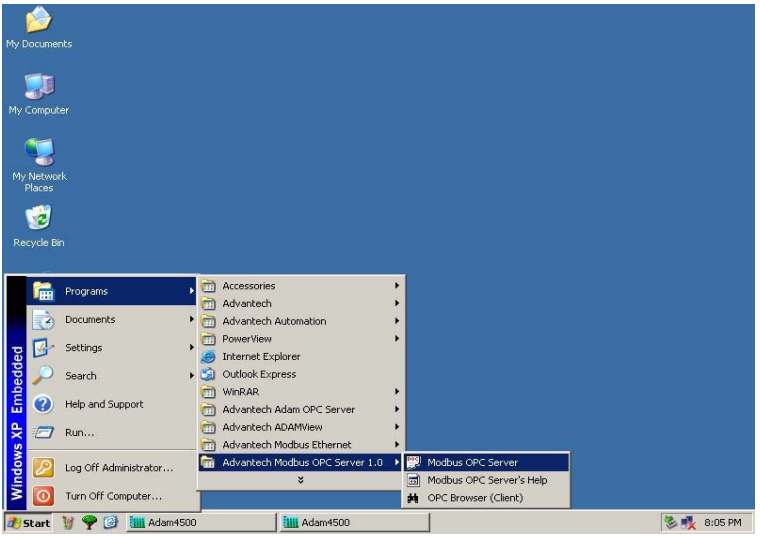
1. Build DEMORS.EXE from DEMORS.PRJ under \ADAM-4500 Series Utility\Source \Example\DemoModbus directory and download DEMORS.EXE onto drive D under root directory. (Put switch into **Initial mode)**


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
ADAM-4500 Series:
COM1
Name Size Type Modified
CONF File Folder 2006/07/25 6:01:00 AM
WEBROOT File Folder 2006/07/25 6:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/07/25 6:01:00 AM
AX8796.COM 13KB MS-DOS A... 2006/07/25 6:01:00 AM
CFGIO.EXE 35KB Application 2006/07/25 6:01:00 AM
DEMORS.EXE 41KB Application 2006/07/25 7:01:00 AM
DEMOTS.EXE 51KB Application 2006/07/25 6:01:00 AM
SOCKETP.EXE 48KB Application 2006/07/25 6:01:00 AM
8 object(s)
185.9 KB(658,432 bytes free)
Directory of D:\
CONF <DIR> 07-25-2006 6:11p
WEBROOT <DIR> 07-25-2006 6:11p
AUTORUN BAT 128 07-25-2006 6:11p
AX8796 COM 13,184 07-25-2006 6:11p
CFGIO EXE 35,328 07-25-2006 6:11p
SOCKETP EXE 48,896 07-25-2006 6:12p
DEMOTS EXE 51,456 07-25-2006 6:19p
DEMORS EXE 41,344 07-25-2006 7:11p
8 file(s) 190,336 bytes
658,432 bytes free

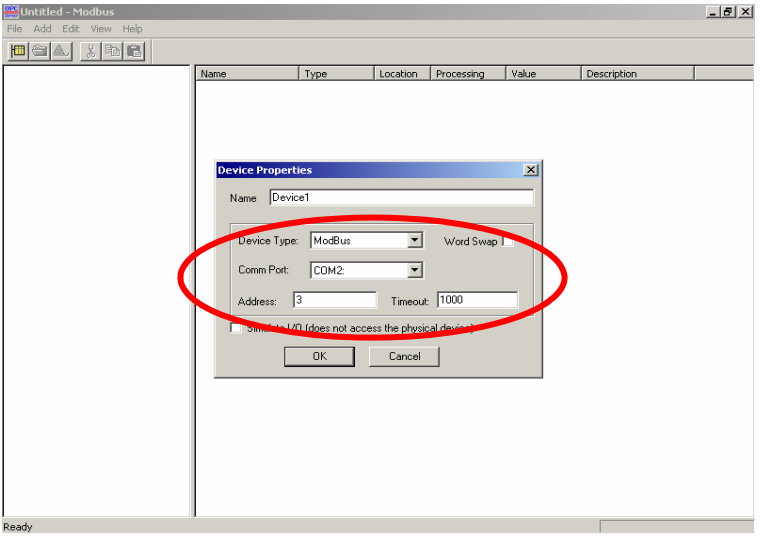
2. Run DEMORS.EXE. ADAM-4500 Series Controller will launch Modbus/RTU master and write DIO value to address 40001.



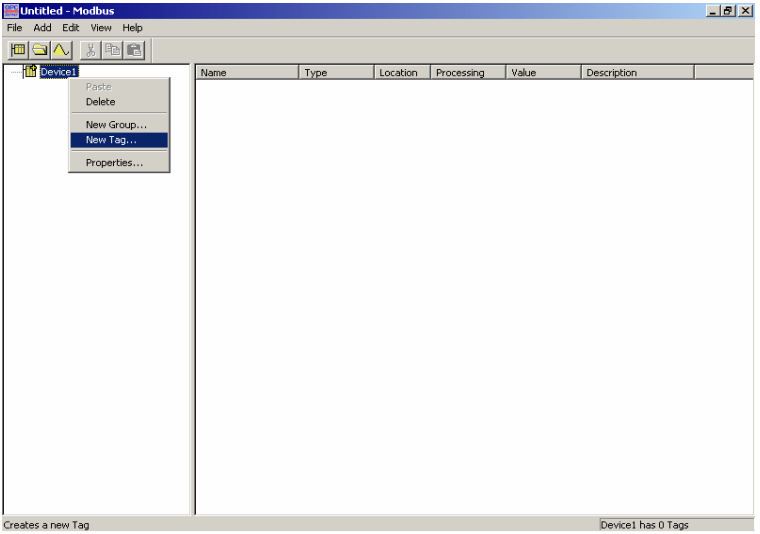
3. Launch Advantech Modbus OPC Server.



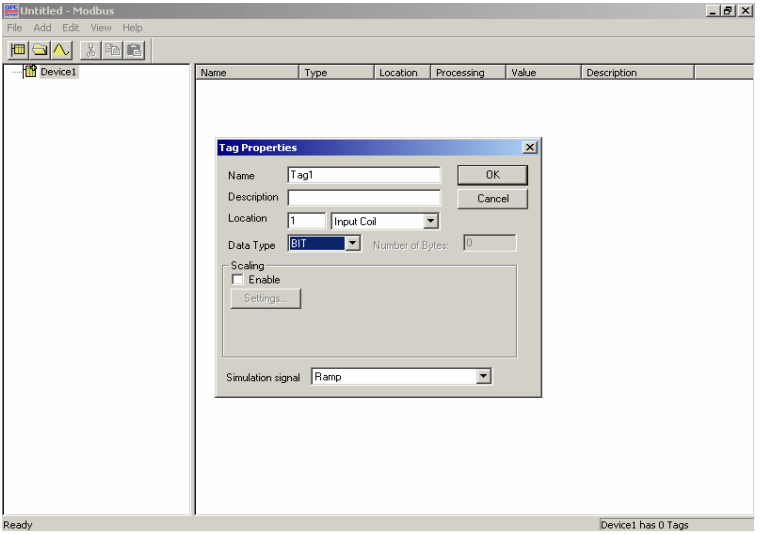
4. Create New Device under Advantech Modbus OPC Server. (Set Address as 3 according to the configuration in DEMORS.C)



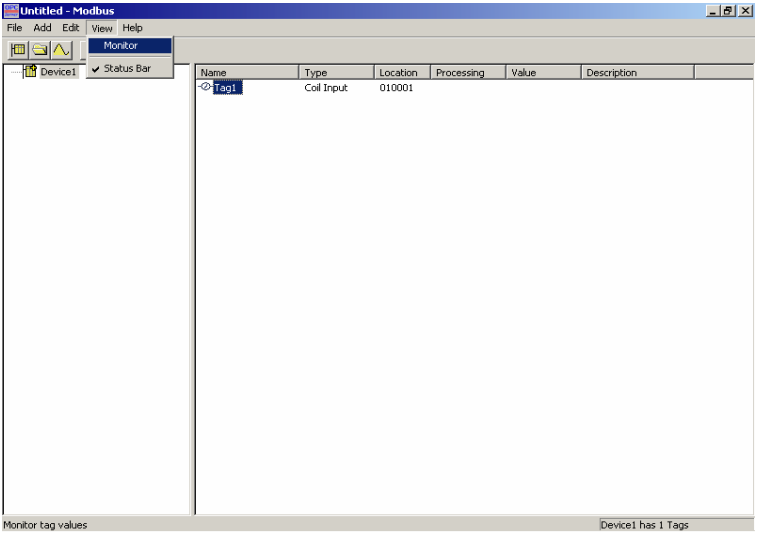
5. Right click on the new device you have just created and choose "New Tag".



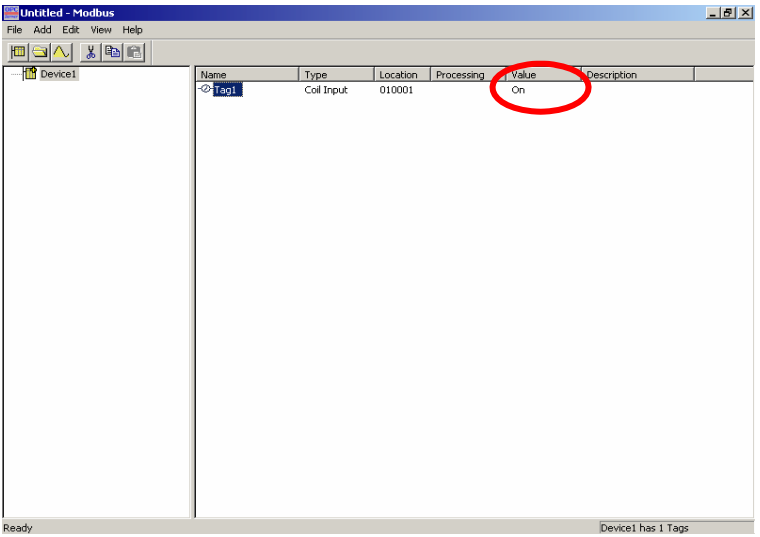
6. Configure the Tag Property.



7. Click “Monitor” under menu “View” to see the value get from ADAM-4500 Series Controller.



8. You can see the DIO value of ADAM-4500 Series Controller.



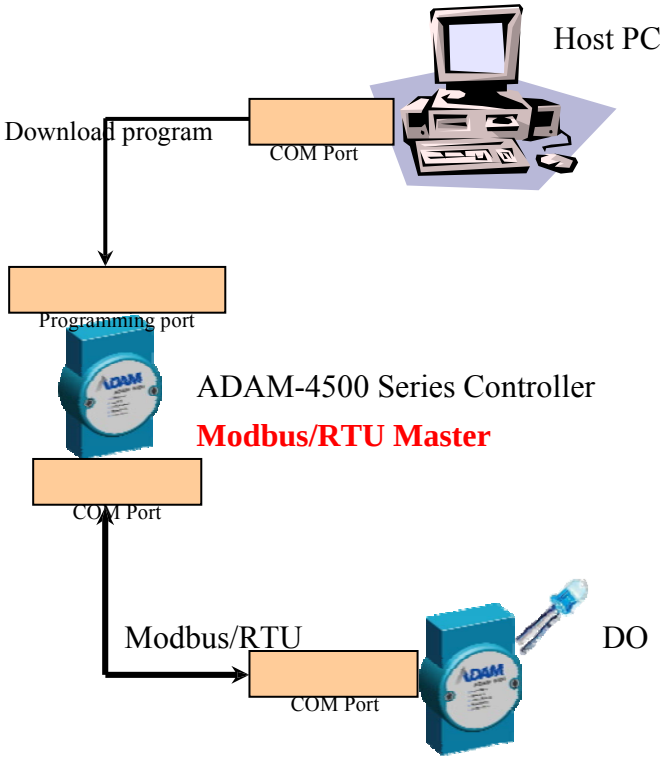
DEMORS.C

```
#include <stdio.h>
#include <dos.h>
#include <time.h>
#include <conio.h>
#include "4500drv.h"
#include "RTU.h"

#define MAXDATASIZE 100
#define sizeofShareMem       10

int count;

void main()
{
        unsigned int Share_Mem[sizeofShareMem];
        char cCh;
        char LSR_State;
        unsigned int tmpcnt, tmpcnt1;
        unsigned long div;

        if(Modbus_COM_Init(COM2, Slave, (unsigned long)9600,
NO_PARITY, DATA8, STOP1)!=0)
        {
                printf("error\n");
                return;
        }

        printf("init success!!\n");

        ADAMRTU_ModServer_Create(3, (unsigned char *)Share_Mem,
sizeof(Share_Mem));
        //      {
        //              printf("err code is %d\n", Error_Code());
        //              return;
        //      }

        printf("server started..\n");

        while(1)
        {
                disable();
                GetDIO(EB50_ID, AllChannels, 0, &div);
```

```
                Share_Mem[0] = div;
                //write DIO status to modbus address 40001
                enable();

                for(tmpcnt=0; tmpcnt<50000; tmpcnt++)    //delay
                {for(tmpcnt1=0; tmpcnt1<8; tmpcnt1++){}}

        }

}
```

4.7 Modbus/RTU Master

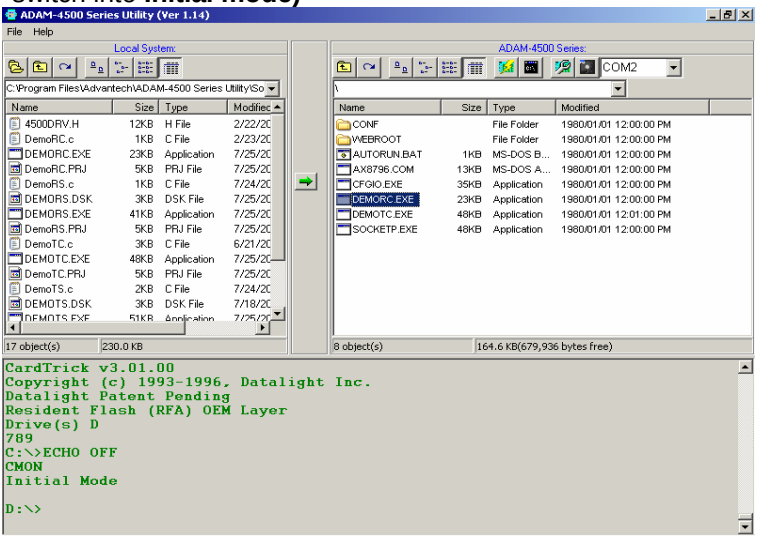
Example program: **DEMORC.EXE**
Source file: **DEMORC.C** under \ADAM-4500 Series Utility\Source \Example\DemoModbus directory
Hardware configuration: see figure below



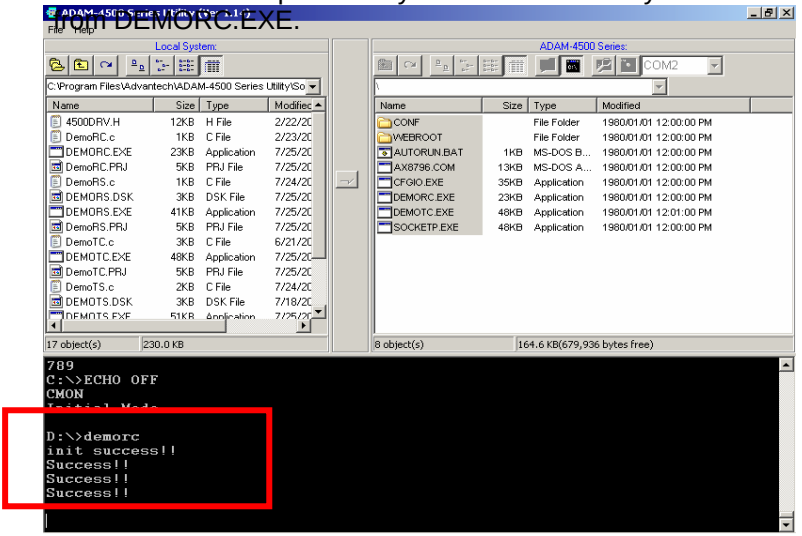
This example uses ADAM-4500 Series Controller as Modbus/RTU master to connect to the ADAM-4056S (support Modbus/RTU protocol and can be a Modbus/RTU slave here), and the DIO value of ADAM-4056S module can be controlled by ADAM-4500 Series

Controller through Modbus/RTU communication.

1. Build DEMORC.EXE from DEMORC.PRJ under \ADAM-4500 Series Utility\Source \Example\DemoModbus directory and download DEMORC.EXE onto drive D under root directory. (Put switch into **Initial mode)**



2. Run DEMORC.EXE and you will find the connection is successful as following figure. You will also find the LEDs of ADAM-4056S periodically switch ON/OFF by the command from DEMORC.EXE.



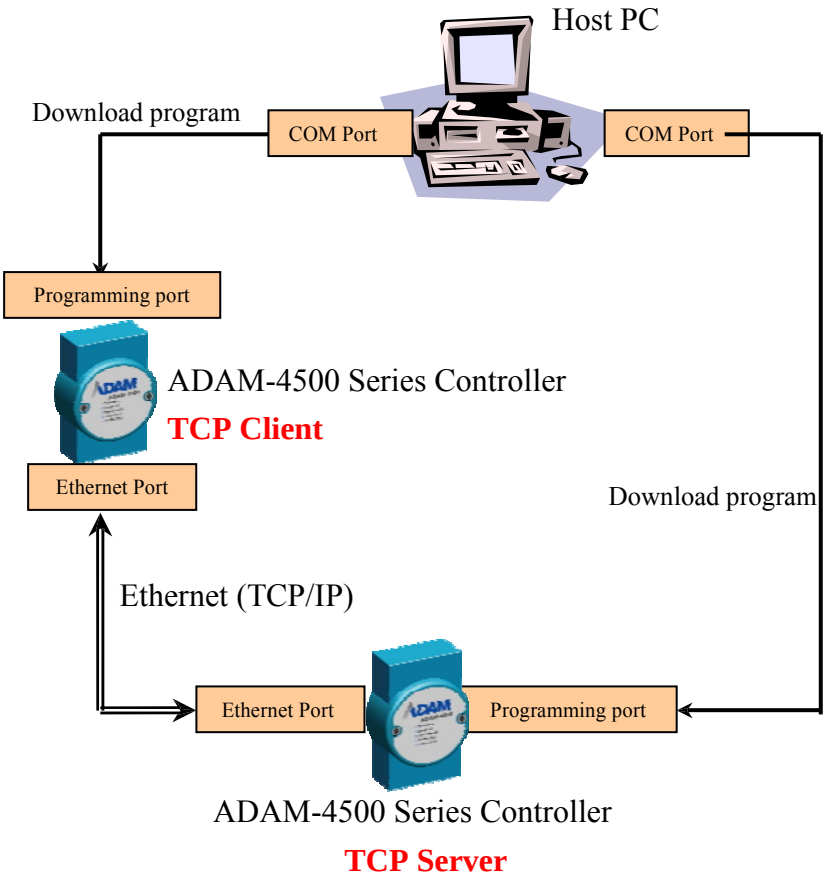
DEMORC.C

```
#include <stdio.h>
#include <dos.h>
#include <time.h>
#include "RTU.h"

#define MAXDATASIZE 100

void main()
{
        unsigned char HostData[MAXDATASIZE];
        int cnt=0;
        unsigned int tmpcnt=0, tmpcnt1=0;

        if(Modbus_COM_Init(COM2, Master, (unsigned long)9600,
NO_PARITY, DATA8, STOP1)!=0)
        {
                printf("error\n");
                return;
        }

        printf("init success!!\n");

        while(1)
        {

                cnt++;
                if(cnt%2==0)
                {
                        HostData[1]=0x0f;
                        HostData[0]=0xff;
                }
                else
                {
                        HostData[1]=0x00;
                        HostData[0]=0x00;
                }
                if(cnt==10)
                        cnt = 0;

                //Set 4056S status
                if(!ADAMRTU_ForceMultiCoils(COM2, 0x02, 0x11, 0x0c,
0x02, HostData))
```

```
                {
                        printf("err code is %d\n", Error_Code());
                        printf("fail send..");
                }
                else
                        printf("Success!!\n");

                for(tmpcnt=0; tmpcnt<50000; tmpcnt++)    //delay
                {for(tmpcnt1=0; tmpcnt1<4; tmpcnt1++){}}

        }

}
```

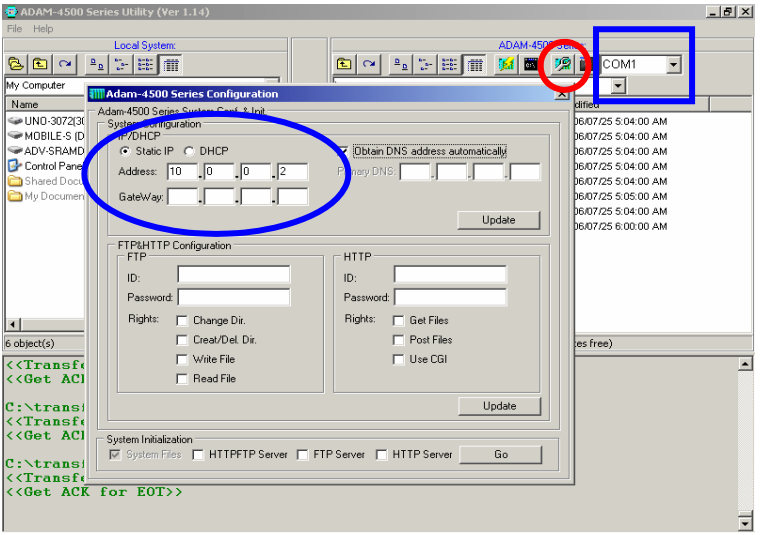
4.8 TCP Server and Client

Example program: **TSERVER.EXE** and **TCLIENT.EXE**
Source file: **TCP_SERVER.C** and **TCP_CLIENT.C** under \ADAM-4500 Series Utility\Source \Example\TCP directory
Hardware configuration: see figure below

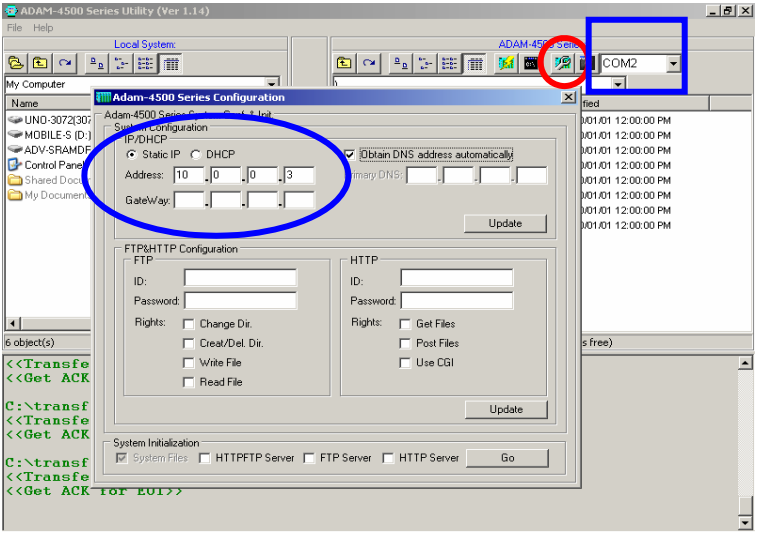


This example uses one ADAM-4500 Series Controller as TCP client to connect to another ADAM-4500 Series Controller (as TCP server). They can exchange data between each other through TCP/IP communication.

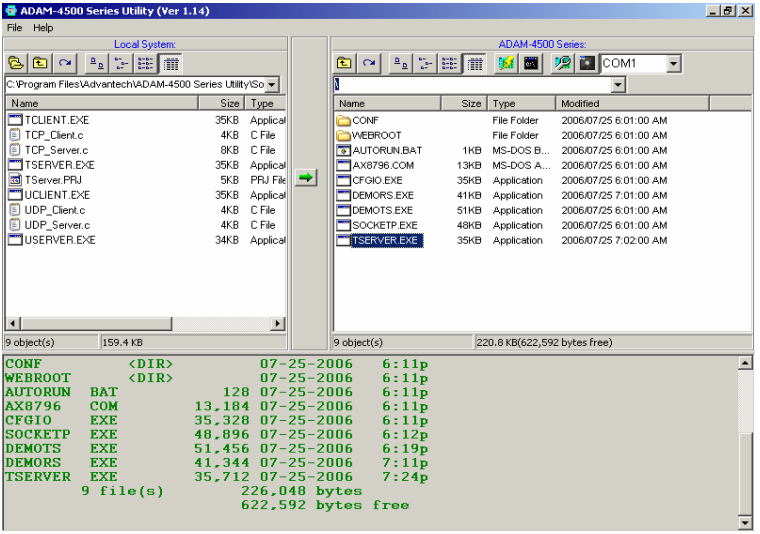
1. Put switch of the two ADAM-4500 Series Controllers into **Initial mode.** Click the "Adam-4500 Configuration" button to set IP address for TCP Server (ADAM-4500 Series Controller)



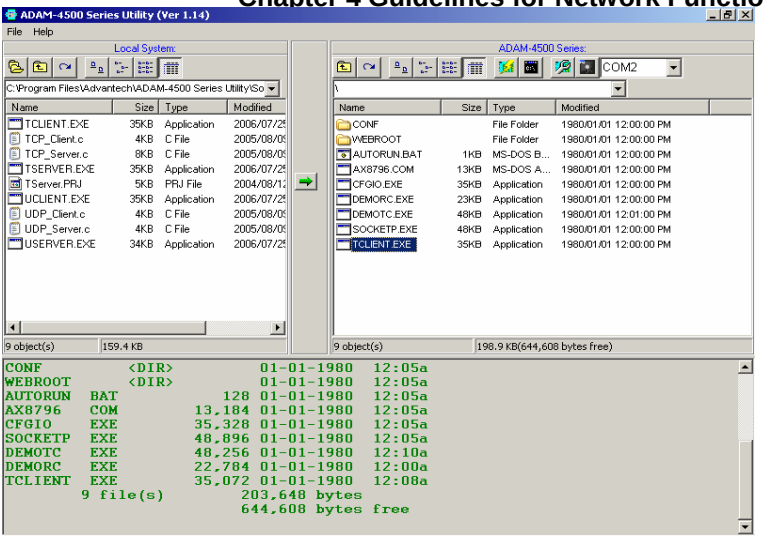
2. Click the "Adam-4500 Configuration" button to set IP address for TCP Client (another ADAM-4500 Series Controller)



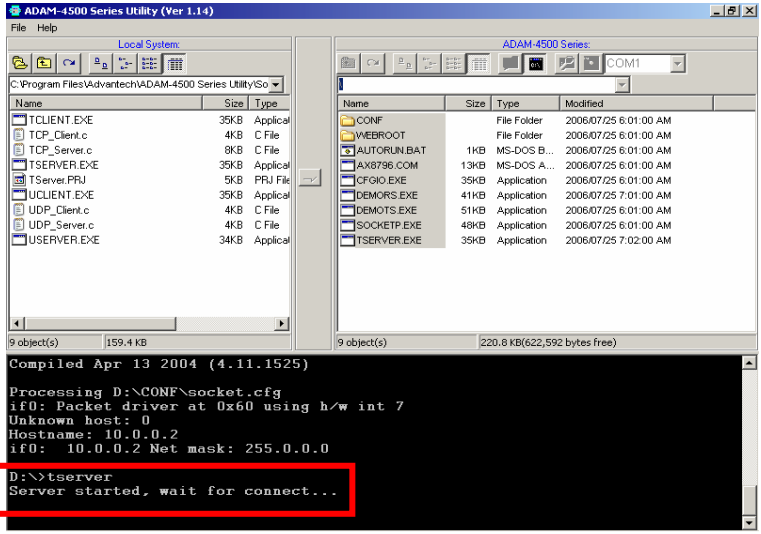
3. Build TSERVER.EXE from TSERVER.PRJ under \ADAM-4500 Series Utility\Source \Example\TCP and download it onto drive D of the ADAM-4500 Series Controller which is the TCP server.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
ADAM-4500 Series:
COM1
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type
TCLIENT.EXE 35KB Applical
TCP_Client.c 4KB C File
TCP_Server.c 8KB C File
TSERVER.EXE 35KB Applical
TServer.PRJ 5KB PRJ File
UCLIENT.EXE 35KB Applical
UDP_Client.c 4KB C File
UDP_Server.c 4KB C File
USERVER.EXE 34KB Applical
9 object(s) 159.4 KB
Name Size Type Modified
CONF File Folder 2006/07/25 6:01:00 AM
WEBROOT File Folder 2006/07/25 6:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/07/25 6:01:00 AM
AX8796.COM 13KB MS-DOS A... 2006/07/25 6:01:00 AM
CFGIO.EXE 35KB Application 2006/07/25 6:01:00 AM
DEMORS.EXE 41KB Application 2006/07/25 7:01:00 AM
DEMOTS.EXE 51KB Application 2006/07/25 6:01:00 AM
SOCKETP.EXE 48KB Application 2006/07/25 6:01:00 AM
TSERVER.EXE 35KB Application 2006/07/25 7:02:00 AM
9 object(s) 220.8 KB(622,592 bytes free)
CONF <DIR> 07-25-2006 6:11p
WEBROOT <DIR> 07-25-2006 6:11p
AUTORUN BAT 128 07-25-2006 6:11p
AX8796 COM 13,184 07-25-2006 6:11p
CFGIO EXE 35,328 07-25-2006 6:11p
SOCKETP EXE 48,896 07-25-2006 6:12p
DEMOTS EXE 51,456 07-25-2006 6:19p
DEMORS EXE 41,344 07-25-2006 7:11p
TSERVER EXE 35,712 07-25-2006 7:24p
9 file(s) 226,048 bytes
622,592 bytes free

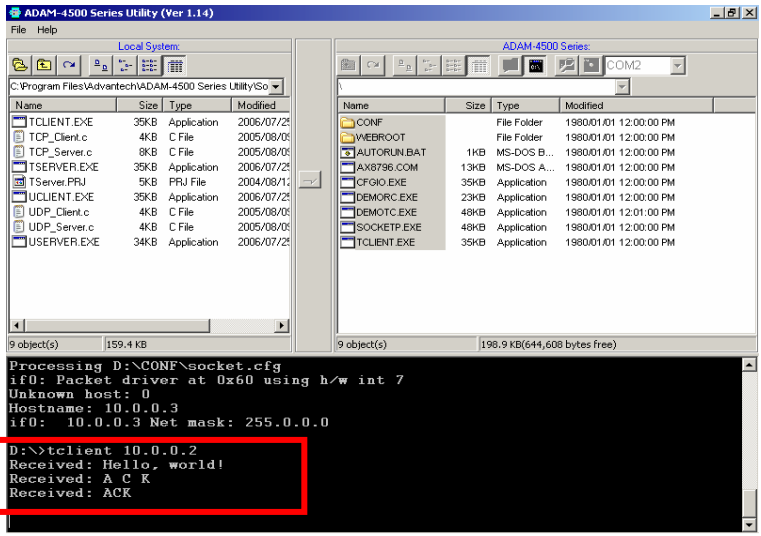
4. Build TCLIENT.EXE from TCLIENT.PRJ under \ADAM-4500 Series Utility\Source \Example\TCP directory and download it onto drive D of the ADAM-4500 Series Controller which is the TCP client.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type Modified
TCLIENT.EXE 35KB Application 2006/07/25
TCP_Client.c 4KB C File 2005/08/0
TCP_Server.c 8KB C File 2005/08/0
TSERVER.EXE 35KB Application 2006/07/25
TServer.PRJ 5KB PRJ File 2004/08/1
UCLIENT.EXE 35KB Application 2006/07/25
UDP_Client.c 4KB C File 2005/08/0
UDP_Server.c 4KB C File 2005/08/0
USERVER.EXE 34KB Application 2006/07/25
9 object(s) 159.4 KB
ADAM-4500 Series:
COM2
\
Name Size Type Modified
CONF File Folder 1980/01/01 12:00:00 PM
WEBROOT File Folder 1980/01/01 12:00:00 PM
AUTORUN.BAT 1KB MS-DOS B... 1980/01/01 12:00:00 PM
AX8796.COM 13KB MS-DOS A... 1980/01/01 12:00:00 PM
CFGIO.EXE 35KB Application 1980/01/01 12:00:00 PM
DEMORC.EXE 23KB Application 1980/01/01 12:00:00 PM
DEMOTC.EXE 48KB Application 1980/01/01 12:01:00 PM
SOCKETP.EXE 48KB Application 1980/01/01 12:00:00 PM
TCLIENT.EXE 35KB Application 1980/01/01 12:00:00 PM
9 object(s) 198.9 KB(644,608 bytes free)
CONF <DIR> 01-01-1980 12:05a
WEBROOT <DIR> 01-01-1980 12:05a
AUTORUN BAT 128 01-01-1980 12:05a
AX8796 COM 13,184 01-01-1980 12:05a
CFGIO EXE 35,328 01-01-1980 12:05a
SOCKETP EXE 48,896 01-01-1980 12:05a
DEMOTC EXE 48,256 01-01-1980 12:10a
DEMORC EXE 22,784 01-01-1980 12:00a
TCLIENT EXE 35,072 01-01-1980 12:08a
9 file(s) 203,648 bytes
644,608 bytes free

5. Put switch of the two ADAM-4500 Series Controllers into **Normal mode** and reboot the two controllers. Run TSERVER.EXE on the ADAM-4500 Series Controller which is the TCP server.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type
TCLIENT.EXE 35KB Applica
TCP_Client.c 4KB C File
TCP_Server.c 8KB C File
TSERVER.EXE 35KB Applica
TServer.PRJ 5KB PRJ File
UCLIENT.EXE 35KB Applica
UDP_Client.c 4KB C File
UDP_Server.c 4KB C File
USERVER.EXE 34KB Applica
9 object(s) 159.4 KB
ADAM-4500 Series:
COM1
Name Size Type Modified
CONF File Folder 2006/07/25 6:01:00 AM
WEBROOT File Folder 2006/07/25 6:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/07/25 6:01:00 AM
AX8796.COM 13KB MS-DOS A... 2006/07/25 6:01:00 AM
CFGIO.EXE 35KB Application 2006/07/25 6:01:00 AM
DEMORS.EXE 41KB Application 2006/07/25 7:01:00 AM
DEMOTS.EXE 51KB Application 2006/07/25 6:01:00 AM
SOCKETP.EXE 48KB Application 2006/07/25 6:01:00 AM
TSERVER.EXE 35KB Application 2006/07/25 7:02:00 AM
9 object(s) 220.8 KB(622,592 bytes free)
Compiled Apr 13 2004 (4.11.1525)
Processing D:\CONF\socket.cfg
if0: Packet driver at 0x60 using h/w int 7
Unknown host: 0
Hostname: 10.0.0.2
if0: 10.0.0.2 Net mask: 255.0.0.0
D:\>tserver
Server started, wait for connect...

6. Run TCLIENT.EXE (add IP address of TCP server which you set in step 1 as parameter) on another ADAM-4500 Series Controller which is the TCP client. You should see the data received.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type Modified
TCLIENT.EXE 35KB Application 2006/07/25
TCP_Client.c 4KB C File 2005/08/05
TCP_Server.c 8KB C File 2005/08/05
TSERVER.EXE 35KB Application 2006/07/25
TServer.PRJ 5KB PRJ File 2004/08/12
UCLIENT.EXE 35KB Application 2006/07/25
UDP_Client.c 4KB C File 2005/08/05
UDP_Server.c 4KB C File 2005/08/05
USERVER.EXE 34KB Application 2006/07/25
9 object(s) 159.4 KB
ADAM-4500 Series:
COM2
Name Size Type Modified
CONF File Folder 1980/01/01 12:00:00 PM
WEBROOT File Folder 1980/01/01 12:00:00 PM
AUTORUN.BAT 1KB MS-DOS B... 1980/01/01 12:00:00 PM
AX8796.COM 13KB MS-DOS A... 1980/01/01 12:00:00 PM
CFGIO.EXE 35KB Application 1980/01/01 12:00:00 PM
DEMORC.EXE 23KB Application 1980/01/01 12:00:00 PM
DEMOTC.EXE 48KB Application 1980/01/01 12:01:00 PM
SOCKETP.EXE 48KB Application 1980/01/01 12:00:00 PM
TCLIENT.EXE 35KB Application 1980/01/01 12:00:00 PM
9 object(s) 198.9 KB(644,608 bytes free)
Processing D:\CONF\socket.cfg
if0: Packet driver at 0x60 using h/w int 7
Unknown host: 0
Hostname: 10.0.0.3
if0: 10.0.0.3 Net mask: 255.0.0.0
D:\>tclient 10.0.0.2
Received: Hello, world!
Received: A C K
Received: ACK

TCP_SERVER.C

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"
#define Errno errno

#define FALSE 0
#define TRUE 1
#define Host_Port 5510
#define Max_Conn 40
#define MAXDATASIZE 100

SOCKET remoteSocket[Max_Conn];
int WaitSocketCount[Max_Conn];
int socketTotal = 0;
int timeoutRelease = FALSE;

void ReleaseClient(int idx_so);

int main(void)
{
   SOCKET Sock_4500, New_Conn;
   struct sockaddr_in Host_addr;
   struct sockaddr_in Client_addr;
   int sin_size;
   int hasConnect, hasMessage;
   int maxSocket, sidx, New_Sidx, numbytes, sidx2;
   char buf[MAXDATASIZE];
   unsigned long pulArgp;
   char *str;
   int tmpcount=1;

   if ((Sock_4500 = socket(AF_INET, SOCK_STREAM, 0)) ==
INVALID_SOCKET)
```

```
    {
        perror("socket");
        exit(1);
    }

    Host_addr.sin_family = AF_INET;
    Host_addr.sin_port = htons(Host_Port);
    Host_addr.sin_addr.s_addr = INADDR_ANY;
    memset(&(Host_addr.sin_zero), 0, 8);

    if (bind(Sock_4500, (struct sockaddr *)&Host_addr, sizeof(struct
sockaddr)) == SOCKET_ERROR)
    {
        perror("bind");
        exit(1);
    }

    pulArgp = 1;
    if(ioctlsocket(Sock_4500, FIONBIO, &pulArgp))
    {
        perror("ioctlsocket");
        exit(1);
    }


    if (listen(Sock_4500, 5) == SOCKET_ERROR)
    {
        perror("listen");
        exit(1);
    }

    hasMessage = FALSE;
    memset(WaitSocketCount, 0, sizeof(WaitSocketCount));
    printf("Server started, wait for connect...\n");
    while(1)
    {
        if (socketTotal > 0)
            hasConnect = Host_WaitForClient(Sock_4500, 0);
        else
            hasConnect = Host_WaitForClient(Sock_4500, 5);
```

```
    if(hasConnect)
    {
      printf("Receive client connect request...\n");
      sin_size = sizeof(struct sockaddr_in);
      if ((New_Conn = accept(Sock_4500, (struct sockaddr *)&Client_addr,
        &sin_size)) == INVALID_SOCKET)
      {
        perror("accept");
        continue;
      }

      if (New_Conn != INVALID_SOCKET)
      {
        if (socketTotal < Max_Conn)
        {
          remoteSocket[socketTotal] = New_Conn;
          New_Sidx = socketTotal;
          socketTotal++;
        }
        else
        {
          if (send(New_Conn, "Connetion full, you are going to be
disconnected!\n", 50, 0) == SOCKET_ERROR)
            perror("send");
          closesocket(New_Conn);
          printf("Connetion full, disconnect client!\n");
        }
      }
      else
        printf("(TCP) Invalid incoming socket!\n");


      str = "Hello, world!\n";
      if (send(remoteSocket[New_Sidx], str, strlen(str), 0) ==
SOCKET_ERROR)
        perror("send");

    }

    if(socketTotal>0)
    {
      for(sidx=0; sidx<socketTotal; sidx++)
      {
        hasMessage = Host_WaitForClient(remoteSocket[sidx], 0);
```

```
        if(hasMessage)
        {
          if((numbytes=recv(remoteSocket[sidx], buf, sizeof(buf), 0)) ==
SOCKET_ERROR)
          {
            ReleaseClient(sidx);
          }
          else
          {

              if(numbytes>0)
                printf("Server receive: %s", buf);

              if(tmpcount%2==0)
              str = "ACK\n"; else
                str = "A C K\n";

              if(numbytes==0)
              {
                      ReleaseClient(sidx);
              }
              else if(send(remoteSocket[sidx], str, strlen(str), 0) ==
SOCKET_ERROR)
              { ReleaseClient(sidx);
              }


              memset(buf, 0, sizeof(buf));
              tmpcount++;
              if(tmpcount>100)
              tmpcount = 1;

              WaitSocketCount[sidx] = 0;
          }
        }
        else
          WaitSocketCount[sidx]++;

        if(WaitSocketCount[sidx]>10000)
        {
          timeoutRelease = TRUE;
```

```
                ReleaseClient(sidx);
            }
        }
    }
}

return 0;
}

int Host_WaitForClient(int WaitSocket, int i_iWaitMilliSec)
{
fd_set FdSet;
struct timeval    waitTime;

FD_ZERO(&FdSet);
FD_SET(WaitSocket, &FdSet);
waitTime.tv_sec    = i_iWaitMilliSec / 1000;
waitTime.tv_usec = (i_iWaitMilliSec % 1000)*1000L;

if (select(0, &FdSet, NULL, NULL, &waitTime) > 0)
    return TRUE;
return FALSE;
}

void ReleaseClient(int idx_so)
{
int sidx, sidx2;

sidx = idx_so;

if(timeoutRelease)
{
    if (send(remoteSocket[sidx], "Connetion timeout, you are going to be
disconnected!\n", 53, 0) == -1)
        perror("send");
}

if(remoteSocket[sidx]!=INVALID_SOCKET)
{
        if(closesocket(remoteSocket[sidx])!=0)
                printf("Release client resource fail!");
}
```

```
    for(sidx2 = sidx; sidx2<= socketTotal-1; sidx2++)
    {
        if(sidx2<socketTotal-1)
        {
            WaitSocketCount[sidx2] = WaitSocketCount[sidx2+1];
            remoteSocket[sidx2] = remoteSocket[sidx2+1];
        }
        else if(sidx2==socketTotal-1)
        {
            WaitSocketCount[sidx2] = 0;
            remoteSocket[sidx2] = NULL;
        }

    }

    socketTotal--;

    if(timeoutRelease)
        printf("Connetion timeout, disconnect client %d!\n", sidx);
    else
        printf("Socket error, disconnect client %d!\n", sidx);

    if(socketTotal==0)
        printf("Wait for client connect...\n");

    timeoutRelease = FALSE;

}
```

TCP_CLIENT.C

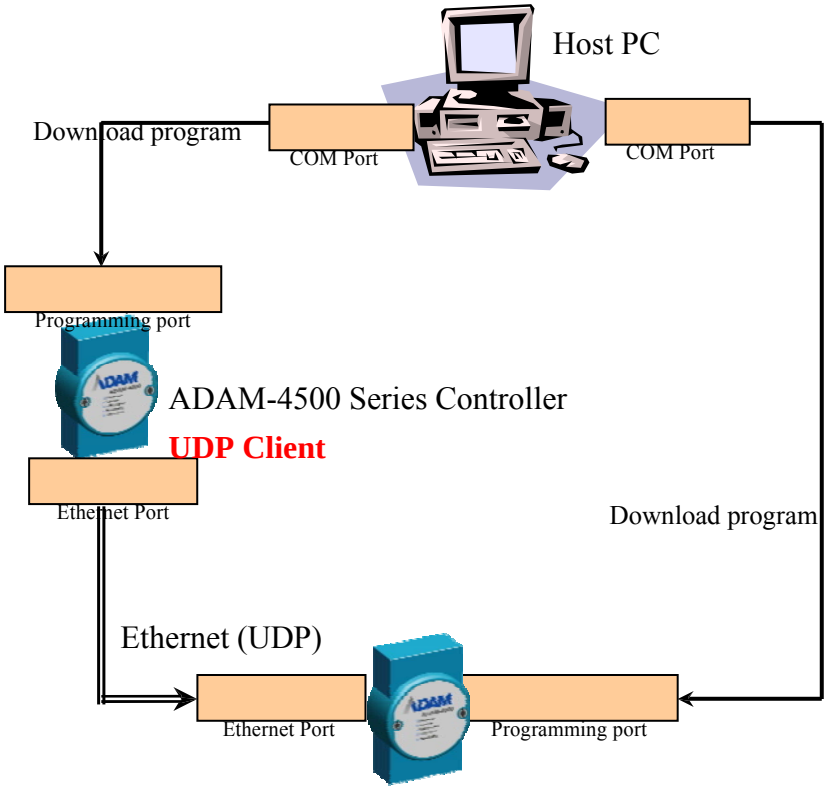
```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"
#define Errno errno

#define Server_Port 5510
#define MAXDATASIZE 100

int main(int argc, char *argv[])
{
   SOCKET SO_4500;
   int numbytes=0;
   char buf[MAXDATASIZE];
   struct hostent *he;
   struct sockaddr_in Server_addr;
   char *str1, *str2, *str;
   int tmpcount=1;

   str1 = "TCP\n";
   str2 = "Client\n";

   if (argc != 2)
   {
      fprintf(stderr,"usage: server hostname\n");
      exit(1);
   }

   if ((he=gethostbyname(argv[1])) == NULL)
   {
      perror("gethostbyname");
      exit(1);
   }
```

```
   if ((SO_4500 = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
INVALID_SOCKET)
   {
      perror("socket");
      exit(1);
   }

   Server_addr.sin_family = AF_INET;
   Server_addr.sin_port = htons(Server_Port);
   Server_addr.sin_addr = *((struct in_addr *)he->h_addr);
   memset(&(Server_addr.sin_zero), 0, 8);


   if (connect(SO_4500, (struct sockaddr *)&Server_addr,
      sizeof(struct sockaddr)) == SOCKET_ERROR)
   {
      perror("connect");
      exit(1);
   }

   while(1)
   {
      if ((numbytes=recv(SO_4500, buf, MAXDATASIZE-1, 0)) ==
SOCKET_ERROR)
      {
         perror("recv");
         exit(1);
      }

      if(numbytes>0)
      {
         printf("Received: %s",buf);

         memset(buf, 0, sizeof(buf));
         if(tmpcount%2==0)
            str = str1;
         else
            str = str2;

         sleep(1);
         if (send(SO_4500, str, strlen(str), 0) == SOCKET_ERROR)
         {
```

```
            perror("send");
            exit(1);
         }
         tmpcount++;
         if(tmpcount>100)
         tmpcount=1;
      }
      else
      {
         closesocket(SO_4500);
         break;
      }
   }
   return 0;
}
```

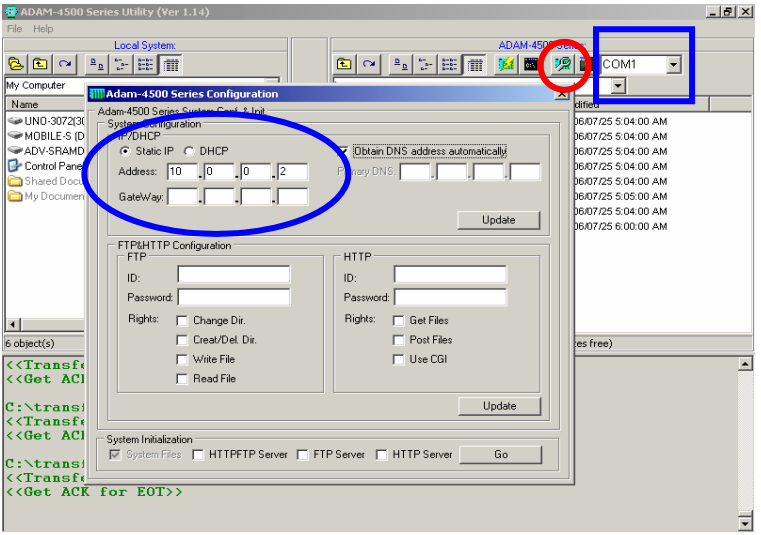
4.9 UDP Connection

Example program: **USERVER.EXE** and **UCLIENT.EXE**
Source file: **UDP_SERVER.C** and **UDP_CLIENT.C** under \ADAM-4500 Series Utility\Source \Example\TCP directory
Hardware configuration: see figure below

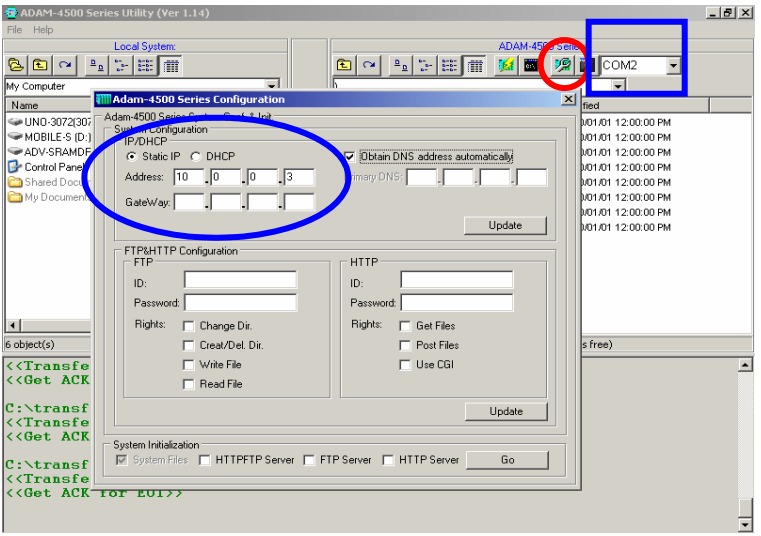


This example uses one ADAM-4500 Series Controller as UDP client to connect to another ADAM-4500 Series Controller (as UDP server). They can exchange data between each other through UDP communication.

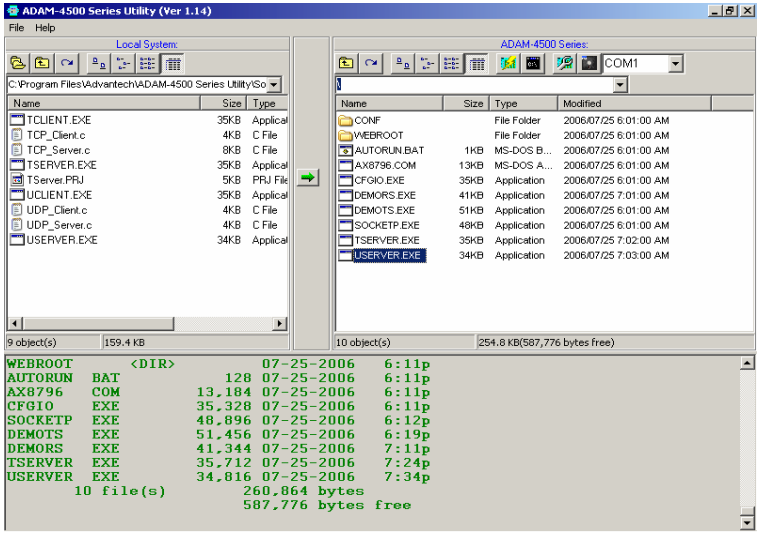
1. Put switch of the two ADAM-4500 Series Controllers into **Initial mode.** Click the "Adam-4500 Configuration" button to set IP address for Server (ADAM-4500 Series Controller)



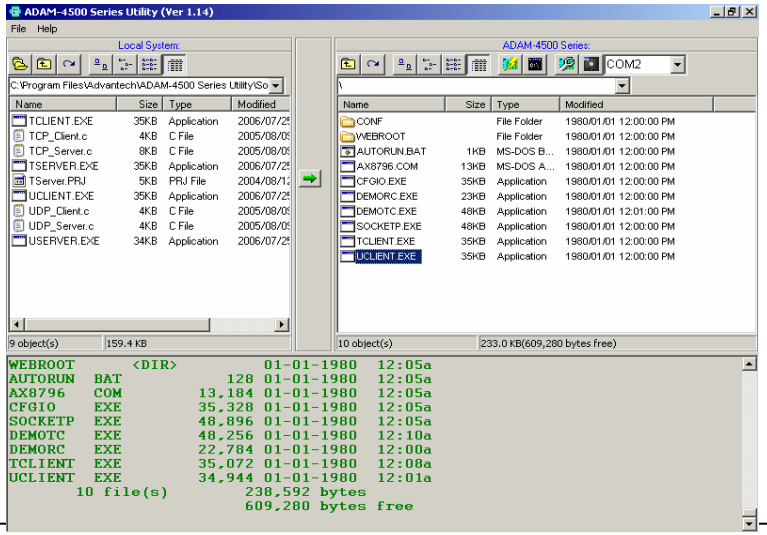
2. Click the "Adam-4500 Configuration" button to set IP address for Client (another ADAM-4500 Series Controller)



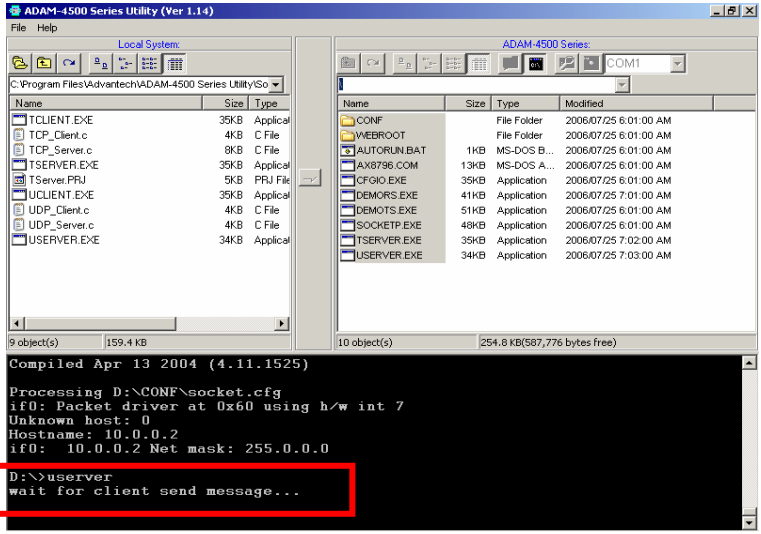
3. Build USERVER.EXE from USERVER.PRJ under \ADAM-4500 Series Utility\Source \Example\TCP and download it onto drive D of the ADAM-4500 Series Controller which is the UDP server.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
ADAM-4500 Series:
COM1
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type
TCLIENT.EXE 35KB Applical
TCP_Client.c 4KB C File
TCP_Server.c 8KB C File
TSERVER.EXE 35KB Applical
TServer.PRJ 5KB PRJ File
UCLIENT.EXE 35KB Applical
UDP_Client.c 4KB C File
UDP_Server.c 4KB C File
USERVER.EXE 34KB Applical
9 object(s) 159.4 KB
Name Size Type Modified
CONF File Folder 2006/07/25 6:01:00 AM
WEBROOT File Folder 2006/07/25 6:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/07/25 6:01:00 AM
AX8796.COM 13KB MS-DOS A... 2006/07/25 6:01:00 AM
CFGIO.EXE 35KB Application 2006/07/25 6:01:00 AM
DEMORS.EXE 41KB Application 2006/07/25 7:01:00 AM
DEMOTS.EXE 51KB Application 2006/07/25 6:01:00 AM
SOCKETP.EXE 48KB Application 2006/07/25 6:01:00 AM
TSERVER.EXE 35KB Application 2006/07/25 7:02:00 AM
USERVER.EXE 34KB Application 2006/07/25 7:03:00 AM
10 object(s) 254.8 KB(587,776 bytes free)
WEBROOT <DIR> 07-25-2006 6:11p
AUTORUN BAT 128 07-25-2006 6:11p
AX8796 COM 13,184 07-25-2006 6:11p
CFGIO EXE 35,328 07-25-2006 6:11p
SOCKETP EXE 48,896 07-25-2006 6:12p
DEMOTS EXE 51,456 07-25-2006 6:19p
DEMORS EXE 41,344 07-25-2006 7:11p
TSERVER EXE 35,712 07-25-2006 7:24p
USERVER EXE 34,816 07-25-2006 7:34p
10 file(s) 260,864 bytes
587,776 bytes free

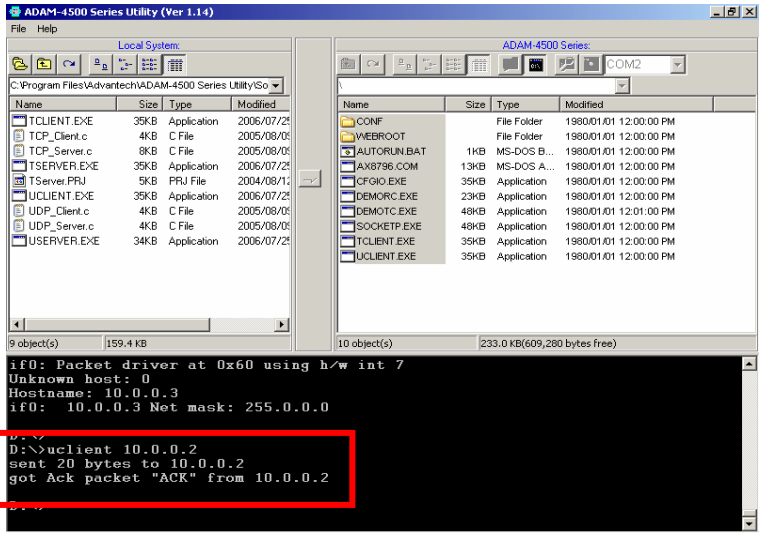
4. Build UCLIENT.EXE from UCLIENT.PRJ under \ADAM-4500 Series Utility\Source \Example\TCP and download it onto drive D of the ADAM-4500 Series Controller which is the UDP client.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
ADAM-4500 Series:
COM2
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type Modified
TCLIENT.EXE 35KB Application 2006/07/25
TCP_Client.c 4KB C File 2005/08/05
TCP_Server.c 8KB C File 2005/08/05
TSERVER.EXE 35KB Application 2006/07/25
TServer.PRJ 5KB PRJ File 2004/08/12
UCLIENT.EXE 35KB Application 2006/07/25
UDP_Client.c 4KB C File 2005/08/05
UDP_Server.c 4KB C File 2005/08/05
USERVER.EXE 34KB Application 2006/07/25
9 object(s) 159.4 KB
Name Size Type Modified
CONF File Folder 1980/01/01 12:00:00 PM
WEBROOT File Folder 1980/01/01 12:00:00 PM
AUTORUN.BAT 1KB MS-DOS B... 1980/01/01 12:00:00 PM
AX8796.COM 13KB MS-DOS A... 1980/01/01 12:00:00 PM
CFGIO.EXE 35KB Application 1980/01/01 12:00:00 PM
DEMORC.EXE 23KB Application 1980/01/01 12:00:00 PM
DEMOTC.EXE 48KB Application 1980/01/01 12:01:00 PM
SOCKETP.EXE 48KB Application 1980/01/01 12:00:00 PM
TCLIENT.EXE 35KB Application 1980/01/01 12:00:00 PM
UCLIENT.EXE 35KB Application 1980/01/01 12:00:00 PM
10 object(s) 233.0 KB(609,280 bytes free)
WEBROOT <DIR> 01-01-1980 12:05a
AUTORUN BAT 128 01-01-1980 12:05a
AX8796 COM 13,184 01-01-1980 12:05a
CFGIO EXE 35,328 01-01-1980 12:05a
SOCKETP EXE 48,896 01-01-1980 12:05a
DEMOTC EXE 48,256 01-01-1980 12:10a
DEMORC EXE 22,784 01-01-1980 12:00a
TCLIENT EXE 35,072 01-01-1980 12:08a
UCLIENT EXE 34,944 01-01-1980 12:01a
10 file(s) 238,592 bytes
609,280 bytes free

5. Put switch of the two ADAM-4500 Series Controllers into **Normal mode** and reboot the two controllers. Run USERVER.EXE on the ADAM-4500 Series Controller which is the UDP server.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type
TCLIENT.EXE 35KB Applica
TCP_Client.c 4KB C File
TCP_Server.c 8KB C File
TSERVER.EXE 35KB Applica
TServer.PRJ 5KB PRJ File
UCLIENT.EXE 35KB Applica
UDP_Client.c 4KB C File
UDP_Server.c 4KB C File
USERVER.EXE 34KB Applica
9 object(s) 159.4 KB
ADAM-4500 Series:
COM1
Name Size Type Modified
CONF File Folder 2006/07/25 6:01:00 AM
WEBROOT File Folder 2006/07/25 6:01:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/07/25 6:01:00 AM
AX8796.COM 13KB MS-DOS A... 2006/07/25 6:01:00 AM
CFGIO.EXE 35KB Application 2006/07/25 6:01:00 AM
DEMORS.EXE 41KB Application 2006/07/25 7:01:00 AM
DEMOTS.EXE 51KB Application 2006/07/25 6:01:00 AM
SOCKETP.EXE 48KB Application 2006/07/25 6:01:00 AM
TSERVER.EXE 35KB Application 2006/07/25 7:02:00 AM
USERVER.EXE 34KB Application 2006/07/25 7:03:00 AM
10 object(s) 254.8 KB(587,776 bytes free)
Compiled Apr 13 2004 (4.11.1525)
Processing D:\CONF\socket.cfg
if0: Packet driver at 0x60 using h/w int 7
Unknown host: 0
Hostname: 10.0.0.2
if0: 10.0.0.2 Net mask: 255.0.0.0
D:\>userver
wait for client send message...

6. Run UCLIENT.EXE (add IP address of TCP server which you set in step 1 as parameter) on another ADAM-4500 Series Controller which is the UDP client. You can see the data sent from server.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
Name Size Type Modified
TCLIENT.EXE 35KB Application 2006/07/25
TCP_Client.c 4KB C File 2005/08/05
TCP_Server.c 8KB C File 2005/08/05
TSERVER.EXE 35KB Application 2006/07/25
TServer.PRJ 5KB PRJ File 2004/08/12
UCLIENT.EXE 35KB Application 2006/07/25
UDP_Client.c 4KB C File 2005/08/05
UDP_Server.c 4KB C File 2005/08/05
USERVER.EXE 34KB Application 2006/07/25
9 object(s) 159.4 KB
ADAM-4500 Series:
COM2
Name Size Type Modified
CONF File Folder 1980/01/01 12:00:00 PM
WEBROOT File Folder 1980/01/01 12:00:00 PM
AUTORUN.BAT 1KB MS-DOS B... 1980/01/01 12:00:00 PM
AX8796.COM 13KB MS-DOS A... 1980/01/01 12:00:00 PM
CFGIO.EXE 35KB Application 1980/01/01 12:00:00 PM
DEMORC.EXE 23KB Application 1980/01/01 12:00:00 PM
DEMOTC.EXE 48KB Application 1980/01/01 12:01:00 PM
SOCKETP.EXE 48KB Application 1980/01/01 12:00:00 PM
TCLIENT.EXE 35KB Application 1980/01/01 12:00:00 PM
UCLIENT.EXE 35KB Application 1980/01/01 12:00:00 PM
10 object(s) 233.0 KB(609,280 bytes free)
if0: Packet driver at 0x60 using h/w int 7
Unknown host: 0
Hostname: 10.0.0.3
if0: 10.0.0.3 Net mask: 255.0.0.0
D:\>uclient 10.0.0.2
sent 20 bytes to 10.0.0.2
got Ack packet "ACK" from 10.0.0.2

UDP_SERVER.C

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"

#define Errno errno

#define FALSE 0
#define TRUE 1
#define Host_Port 5510
#define MAXBUFLEN 100

int main(void)
{
   SOCKET Host_Sock;
   struct sockaddr_in Host_addr;
   struct sockaddr_in Client_addr;
   int hasMessage = FALSE;
   unsigned long pulArgp;
   char buf[MAXBUFLEN];
   int addr_len, numbytes;
   char* ackmsg = "ACK";

   if ((Host_Sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP))
== INVALID_SOCKET)
   {
      perror("socket");
      exit(1);
   }

   Host_addr.sin_family = AF_INET;

   Host_addr.sin_port = htons(Host_Port);
   Host_addr.sin_addr.s_addr = INADDR_ANY;
   memset(&(Host_addr.sin_zero), 0, 8);
```

```
   if (bind(Host_Sock, (struct sockaddr *)&Host_addr, sizeof(struct
sockaddr)) == SOCKET_ERROR)
   {
      perror("bind");
      exit(1);
   }

   pulArgp = 1;
   if(ioctlsocket(Host_Sock, FIONBIO, &pulArgp))
   {
      perror("ioctlsocket");
      exit(1);
   }

   printf("wait for client send message...\n");

   while(1)
   {

      hasMessage = Host_WaitForMessage(Host_Sock, 0);

      if(hasMessage)
      {
         addr_len = sizeof(struct sockaddr);
         if ((numbytes = recvfrom( Host_Sock, buf, sizeof(buf), 0,
            (struct sockaddr *)&Client_addr, &addr_len)) ==
     SOCKET_ERROR)
         {
            perror("recvfrom");
            if (errno == EWOULDBLOCK)
            printf("EWOULDBLOCK");
            break;
         }
         buf[numbytes] = 0;
         printf("got packet \"%s\" from %s\n", buf,
        inet_ntoa(Client_addr.sin_addr));

         if ((numbytes=sendto(Host_Sock, ackmsg, strlen(ackmsg), 0,

            (struct sockaddr *)&Client_addr, sizeof(struct sockaddr))) ==
SOCKET_ERROR)
```

```
            {
                perror("sendto");
                break;
            }
        }
    }

    closesocket(Host_Sock);
    return 0;
}

int Host_WaitForMessage(int serverSocket, int i_iWaitMilliSec)
{
    fd_set FdSet;
    struct timeval    waitTime;

    FD_ZERO(&FdSet);
    FD_SET(serverSocket, &FdSet);
    waitTime.tv_sec    = i_iWaitMilliSec / 1000;
    waitTime.tv_usec = (i_iWaitMilliSec % 1000)*1000L;

    if (select(0, &FdSet, NULL, NULL, &waitTime) > 0)
        return TRUE;
    return FALSE;
}
```

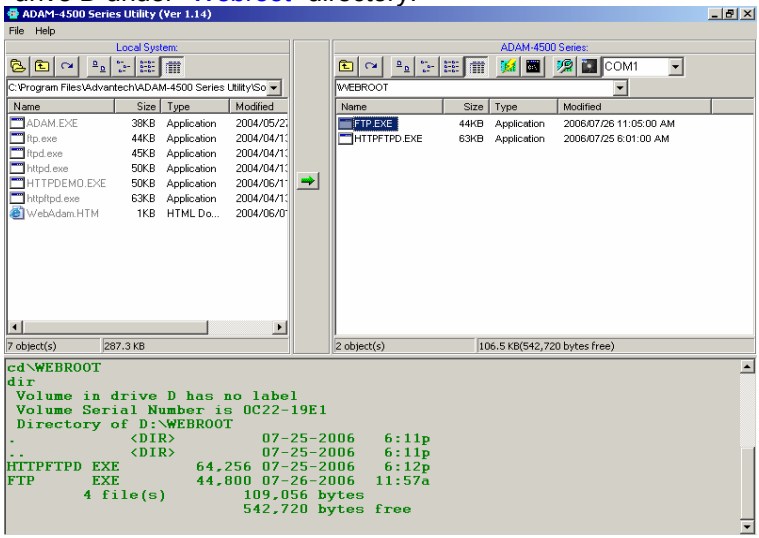
UDP_CLIENT.C

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _MSC_VER
#include <malloc.h>
#else
#include <mem.h>
#endif
#include <string.h>
#include <conio.h>
#include <errno.h>
#include "socket.h"
#define Errno errno
#define BufferSize    100
#define Host_Port 5510

int main(int argc, char *argv[])
{
   SOCKET SO_4500;
   struct sockaddr_in Server_addr;
   struct  sockaddr_in  From_Addr;
   struct hostent *he;
   char buf[BufferSize];
   int numbytes;
   unsigned int From_Size;
   char* msg = "UDP Client Conneted!";

   if (argc != 2)
   {
      fprintf(stderr,"usage: uclient xxx.xxx.xxx.xxx\n");
      exit(1);
   }

   if ((he=gethostbyname(argv[1])) == NULL)
   {
      perror("gethostbyname");
      exit(1);
   }

   if ((SO_4500 = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) ==
INVALID_SOCKET)
   {
      perror("socket");
```

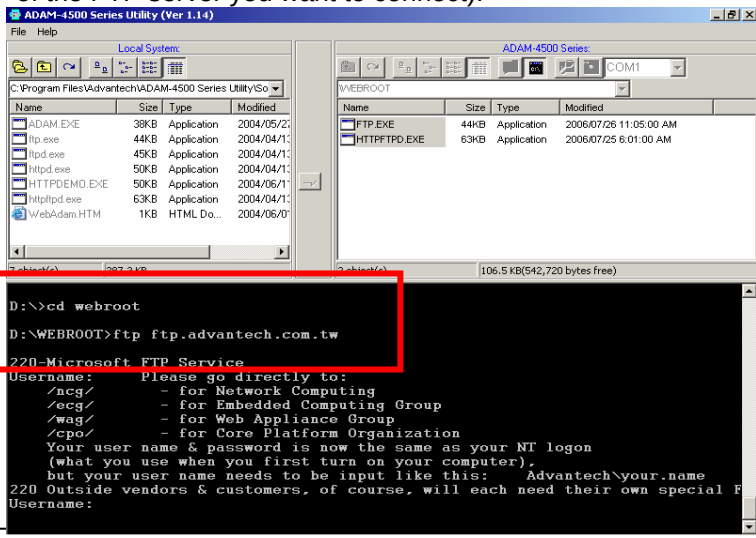
```
        exit(1);
    }

    Server_addr.sin_family = AF_INET;
    Server_addr.sin_port = htons(Host_Port);
    Server_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(&(Server_addr.sin_zero), 0, 8);

    if ((numbytes=sendto(SO_4500, msg, strlen(msg), 0,
        (struct sockaddr *)&Server_addr, sizeof(struct sockaddr))) ==
SOCKET_ERROR)
    {
        perror("sendto");
        exit(1);
    }

    printf("sent %d bytes to %s\n", numbytes,
inet_ntoa(Server_addr.sin_addr));

    From_Size = sizeof(From_Addr);
    if ((numbytes = recvfrom( SO_4500, buf, sizeof(buf), 0,
        (struct sockaddr *)&From_Addr, &From_Size)) == -1)
    {
        perror("recvfrom");
        exit(1);
    }
    buf[numbytes] = 0;
    printf("got Ack packet \"%s\" from %s\n", buf,
inet_ntoa(From_Addr.sin_addr));

    closesocket(SO_4500);

    return 0;
}
```

4.10 FTP Client

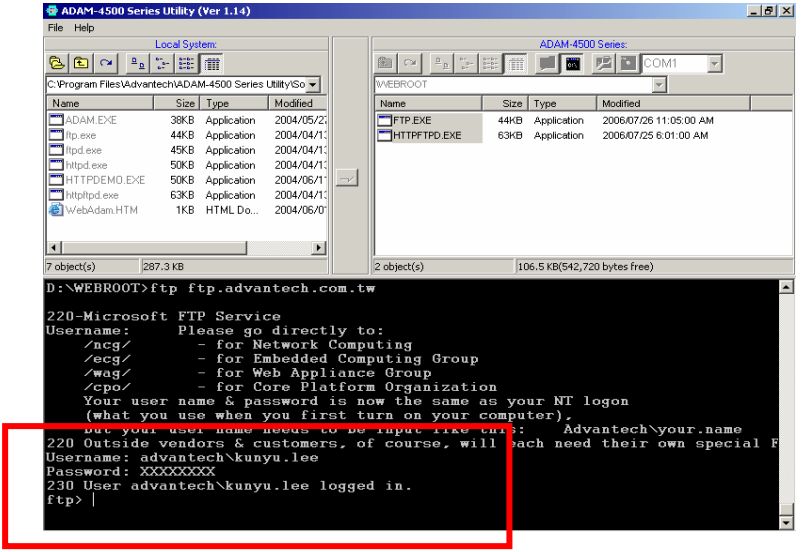
1. Put switch into **Initial mode.** Download FTP.EXE under \ADAM-4500 Series Utility\Source\ Drive_D\Extension_files\WebRoot onto drive D under "Webroot" directory.



2. Put switch into **Normal mode** and reboot. Type **"cd webroot"** to enter "Webroot" directory. Run FTP.EXE (add the FTP address of the FTP server you want to connect).


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
ADAM-4500 Series:
COM1
\WEBROOT
FTP.EXE 44KB Application 2006/07/26 11:05:00 AM
HTTPFTPD.EXE 63KB Application 2006/07/25 6:01:00 AM
106.5 KB(542,720 bytes free)
D:\>cd webroot
D:\WEBROOT>ftp ftp.advantech.com.tw
220-Microsoft FTP Service
Username: Please go directly to:
/ncg/ - for Network Computing
/ecg/ - for Embedded Computing Group
/wag/ - for Web Appliance Group
/cpo/ - for Core Platform Organization
Your user name & password is now the same as your NT logon
(what you use when you first turn on your computer),
but your user name needs to be input like this: Advantech\your.name
220 Outside vendors & customers, of course, will each need their own special F
Username:

3. Login FTP server by typing your username and password. Then you can start to enter FTP command to use the FTP server.


ADAM-4500 Series Utility (Ver 1.14)
File Help
Local System:
ADAM-4500 Series:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
\WEBROOT
COM1
Name Size Type Modified
ADAM.EXE 38KB Application 2004/05/2
ftp.exe 44KB Application 2004/04/1
ftpd.exe 45KB Application 2004/04/1
httpd.exe 50KB Application 2004/04/1
HTTPDEMO.EXE 50KB Application 2004/06/1
httpftpd.exe 63KB Application 2004/04/1
WebAdam.HTM 1KB HTML Do... 2004/06/0
FTP.EXE 44KB Application 2006/07/26 11:05:00 AM
HTTPFTPD.EXE 63KB Application 2006/07/25 6:01:00 AM
7 object(s)
287.3 KB
2 object(s)
106.5 KB(542,720 bytes free)
D:\WEBROOT>ftp ftp.advantech.com.tw
220-Microsoft FTP Service
Username: Please go directly to:
/ncg/ - for Network Computing
/ecg/ - for Embedded Computing Group
/wag/ - for Web Appliance Group
/cpo/ - for Core Platform Organization
Your user name & password is now the same as your NT logon
(what you use when you first turn on your computer),
Advantech\your.name
220 Outside vendors & customers, of course, will each need their own special F
Username: advantech\kunyu.lee
Password: XXXXXXXX
230 User advantech\kunyu.lee logged in.
ftp>

5

Programming and Function Library

5.1 Introduction

User-designed ADAM-4500 Series Controller application programs make use of ADAM-4500 Series library functions. To make the most efficient use of ADAM-4500 Series Controller's memory space, the ADAM-4500 Series function library has been separated into 10 smaller libraries. Therefore, a user can link only those libraries needed
to run his application, and only those libraries will be included in the compiled executable. The smaller the linked libraries, the smaller the compiled executable will be.

5.1.1 Programming detail about the ADAM-4500 Series Controller

The operating system of ADAM-4500 Series Controller is ROM-DOS, which is a MS-DOS equivalent system. It allows users to run application programs written in assembly language as well as high- level languages such as C or C++. Certainly, there will be some limitations when running application programs in the ADAM-4500
Series Controller. In order to build successful applications, please keep the following limitations and concerns in mind.

5.1.2 Mini BIOS functions

The ADAM-4500 Series Controller provides four serial communication ports including programming port for connecting peripherals, so the mini BIOS of ADAM-4500 Series Controller only provides 10 function calls. Since the user's program cannot use other BIOS function calls, the ADAM-4500 Series Controller may not work as intended. Additionally, certain language compilers such as QBASIC directly call BIOS functions that are not executable in ADAM-4500 Series Controller. The ADAM-4500 Series Controller mini BIOS function calls are listed in the following table.

Function	Sub-function	Task
07h		186 or greater cd-processor esc instruct
10h	0eh	TTY Clear output
11h		Get equipment
12h		Get memory size
15h	87h	Extended memory read
	88h	Extended memory size
	c0h	PS/2 or AT style A20 Gate table
16h	0	Read TTY char
	1	Get TTY status
	2	Get TTY flags
18h		Print "Failed to BOOT ROM-DOS" message
19h		Reboot system
1ah	0	Get tick count
	1	Set tick count
	2	Get real time clock
	3	Set real time clock
	4	Get data
	5	Set data
1ch		Timer tick

Table 5-1: ADAM-4500 Series Controller mini BIOS function calls

5.1.3 Converting program codes

The ADAM-4500 Series Controller has an 80188 CPU. Therefore, programs downloaded into its flash ROM must be converted into
80186 or 80188 compatible codes firstly, and the floating point operation must be set to Emulation Mode. For example, if you develop
the application program in Borland C, you will compile the program as following picture.

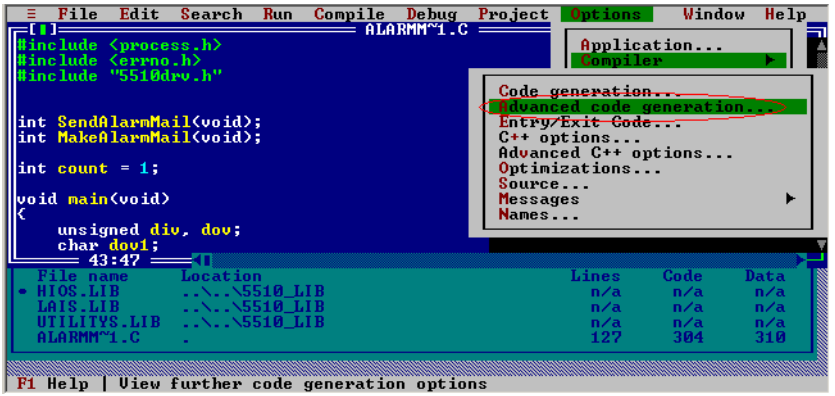


Figure 5-1: Select “Advanced code generation”

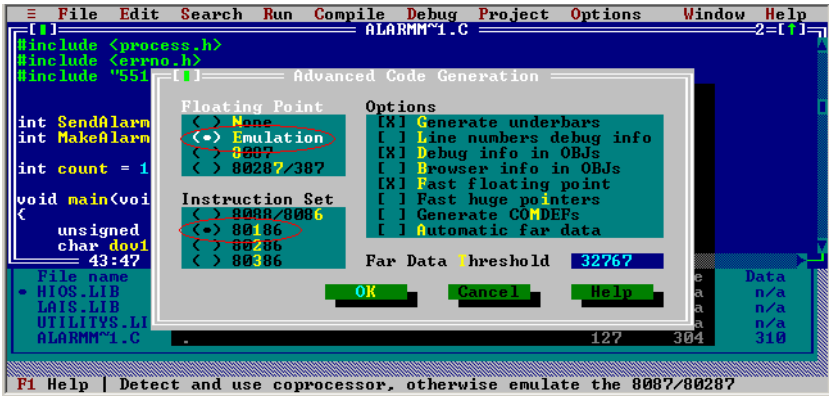


Figure 5-2: Select “Emulation” and “80186” settings

5.1.4 Libraries Sized for Different Memory Modes

The ADAM-4500 Series function libraries support four memory models: SMALL, MEDIUM, COMPACT and LARGE. You can use library files sized according to your memory model. For example, if you use ***small*** model you can link UTILITYS.LIB and LIOS.LIB to implement system and low speed I/O module access functions. On the other hand, if you use ***large*** model, you can link UTILITYL.LIB and LIOL.LIB.

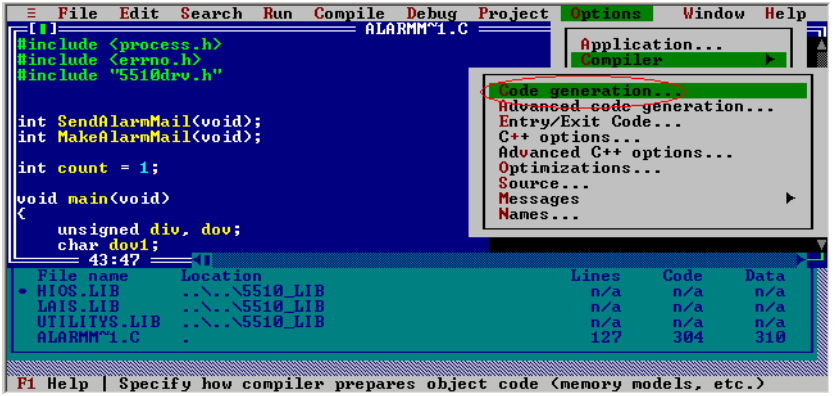


Figure 5-3: Select “Code generation”

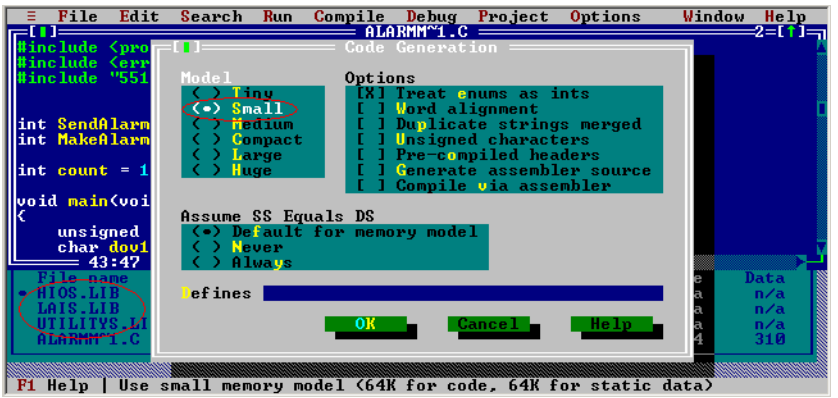


Figure 5-4: Select “Small” Model while using *S.LIB

5.1.5 Limitations

Certain critical files are always kept in flash ROM, such as operating system, BIOS, and monitoring files. The ADAM-4500 Series Controller provides an additional 1MB flash disk as drive D. There is up to
960KB free space for user’s application program. As some additional system files and network utilities for ADAM-4500 Series Controller is distributed on drive D, the free space for user’s application program should be less than 960KB. Besides, there are 256KB flash memory and up to 384KB battery SRAM for user’s applications which can be accessed by function library.

Warning: The free space of flash disk is not suitable for frequently creating and deleting files such as periodic data logging application because the DOS FAT file system is probably destroyed by critical operations while the disk is almost full. The better way is to take the operations on battery backup SRAM.

5.1.6 Programming the watchdog timer

The ADAM-4500 Series Controller is equipped with a watchdog timer function that resets the CPU or generates an interrupt if processing comes to a standstill for any reason. This feature increases system reliability in industrial standalone and unmanned environments.

If you decide to use the watchdog timer, you must write a function call to enable it. When the watchdog timer is enabled, it must be cleared by the application program at intervals of less than 1.6 seconds. If it is not cleared at the required time intervals, it will activate and reset the CPU, or generate a NMI (Non-Maskable Interrupt). You can use a function call in your application program to clear the watchdog timer. At the end of your program, you still need a function call to disable the watchdog timer.

5.2 Category of Function Libraries

ADAM-4500 Series Controller has 10 categories of function libraries as following:

- Category A. System Functions (UTILITY*.LIB)
- Category B. I/O Module Functions (HIO*.LIB)
- Category C. Communication Functions (A4_COMM*.LIB)
- Category D. MODBUS/RTU Functions (RTU*.LIB)
- Category E. MODBUS/TCP Functions (MBTCP*.LIB)
- Category F. Socket Functions (SOCKET*.LIB)
- Category G. HTTP Functions (CGI_LIB*.LIB)

Note 1: These function libraries support Borland C 3.0 for DOS only.
Note 2: Please include all necessary ADAM-4500 Series function libraries in your project file.

5.2.1 Category A: System Functions (UTILITY*.LIB)

adv_printf()
ADAMdelay
Get_BoardID
Get_NodeID
GetRTCtime
SetRTCtime
LED_init
LED_OFF
LED_ON
EraseSector
ProgramByte
ProgramSector
read_mem
Get_SysMem
Set_SysMem
Get_NVRAM_Size
Set_NVRAM_Size
write_backup_ram
read_backup_ram
Timer_Init
Timer_Reset
Timer_Set
Release_All
tmArriveCnt

WDT_clear
WDT_disable
WDT_enable
display_inti()
display_digit()
BatteryStatus()
Ver()

5.2.2 Category B: I/O Module Functions (HIO*.LIB)

InitDIFilter()
GetDIO()
SetDIO()
Ver_HIOLib()

5.2.3 Category C: Communication Functions: (A4_COMM*.LIB)

SIO_Open()
SIO_Close()
SIO_SetState()
SIO_RecvBytes()
SIO_SendBytes()
SIO_GetAvaiRecvBytes()
SIO_GetAvaiSendBuf()
SIO_PurgeBuf()
SIO_MakeCheckSum()
SIO_MakeCRC16()
SIO_Carrier()
SIO_ClearBreak()
SIO_SetBreak()
SIO_GetLineStatus()
SIO_SetLineParams()
SIO_GetModemStatus()
SIO_LowerRaise_RTS_DTR()
SIO_ModemInitial()
SIO_ModemAutoanswer()
SIO_ModemCommand()
SIO_ModemCommand_State()
SIO_ModemDial()
SIO_ModemHandup()
Ver_COMLib()

5.2.4 Category D: MODBUS/RTU Functions (RTU*.LIB)

Modbus_COM_Init()
Modbus_COM_Release()
Error_Code()
ADAMRTU_ForceMultiCoils()
ADAMRTU_ForceSingleCoil()
ADAMRTU_PresetMultiRegs()
ADAMRTU_PresetSingleReg()
ADAMRTU_ReadCoilStatus()
ADAMRTU_ReadHoldingRegs()
ADAMRTU_ReadInputRegs()
ADAMRTU_ReadInputStatus()
ADAMRTU_ModServer_Create()
Ver_RTU_Mod()

5.2.5 Category E: MODBUS/TCP Functions (MBTCP*.LIB)

Ver_TCP_Mod()

Modbus TCP Client Functions:
ReturnErr_code()
ADAMTCP_Connect()
ADAMTCP_Disconnect()
ADAMTCP_ForceMultiCoils()
ADAMTCP_ForceSingleCoil()
ADAMTCP_PresetMultiRegs()
ADAMTCP_PresetSingleReg()
ADAMTCP_ReadCoilStatus()
ADAMTCP_ReadHoldingRegs()
ADAMTCP_ReadInputRegs()
ADAMTCP_ReadInputStatus()

Modbus TCP Server Functions:
ADAMTCP_ModServer_Create()
ADAMTCP_ModServer_Update()
ADAMTCP_ModServer_Release()

5.2.6 Category F: Socket Functions (SOCKET*.LIB)

Socket function:

accept ()
bind ()
closesocket ()
connect ()
ioctlsocket ()
getpeername ()
getsockname ()
getsockopt ()
htonl ()
htons ()
inet_addr ()
inet_ntoa ()
listen ()
ntohl ()
ntohs ()
recv ()
recvfrom ()
select ()
send ()
sendto ()
setsockopt ()
shutdown ()
socket ()

Database function:

gethostbyaddr()
gethostbyname()
gethostname ()
getservbyport()
getservbyname()
getprotobynumber()
getprotobyname()

5.2.7 Category G: HTTP Functions (CGI_LIB*.LIB)

Socket function:

HttpRegister()
HttpDeRegister()
HttpGetData()
HttpSendData()
HttpSubmitFile()
HttpGetStatus()
HttpGetVersion()
GetStackPointer()
GetStackSegment()
SetStackPointer()
SetStackSegment()

5.3 Function Library Description

5.3.1 System Functions (UTILITY*.LIB)

adv_printf

Syntax:
void adv_printf(char *pFormat, ...);

Description:
Print string to console. This function has the same usage as printf() function. However, it has lower priority to be executed.

Parameter
The same as printf() of standard Borland C 3.0 library function.

Return value:
None.

Example:
```
#include "4500drv.h"
void main(void)
{
        adv_printf("Hello, this is for test.");
}
```

Remarks:
If printf() function is put within while loop such as Modbus/RTU server function, it will decrease the performance of server function due to higher priority of printf(). So it is strongly recommended that uses
adv_printf() instead, which has lower priority than printf().

ADAMdelay

Syntax:
void ADAMdelay(unsigned short msec)

Description:
Delays program operation by a specified number of milliseconds.

Parameter	Description
msec	From 0 to 65535.

Return value:
None.

Example:

```
#include "4500drv.h"
void main(void)
{
    /* codes placed here by user */
    ADAMdelay(1000); /* delay 1 sec. */
    /* codes placed here by user */
}
```

Remarks:
ADAMDelay will possibly decrease the performance so it is recommended to use for loop instead.

Get_BoardID

Syntax:
unsigned char Get_BoardID(void)

Description:
Gets the ID number of EB50 (built-in I/O board).

Parameter
None.

Return value:
The return value is the ID number of EB50 (built-in I/O board).

Remarks:
None.

Get_NodeID

Syntax:
unsigned char Get_NodeID(void)

Description:
Gets the ID number of the ADAM-4500 Series Controller.

Parameter
None.

Return value:
The ID number of the ADAM-4500 Series Controller.

Example:

```
#include "4500drv.h"
void main(void)
{
        int i, j;
        unsigned char     mID, found;

  adv_printf("\n Welcome to ADAM4500 PC-Based Controller");
  adv_printf("\n Scan I/O module ...");
  adv_printf("\n ADAM4500 NodeID = %02Xh", Get_NodeID() );

  /* Scan ADAM4500 Slot IO Module */

  mID = Get_BoardID();
  found=0;
        if( mID == EB50_ID)
        {
                adv_printf("\n Slot = EB50");
                found=1;
        }
  if(found == 0) adv_printf("\n Slot = None installed");
}
```

Remarks :
None

GetRTCtime
SetRTCtime

Syntax:
unsigned char GetRTCtime(unsigned char Time)
void SetRTCtime(unsigned char Time,unsigned char data)

Description:
GetRTCtime: Reads Real-Time Clock chip timer. A user can activate a program on the date desired.
SetRTCtime: Sets date and time of the real-time clock.

Parameter	Description	
Time	RTC_sec	the second
	RTC_min	the minute
	RTC_hour	the hour
	RTC_day	the day
	RTC_week	day of the week
	RTC_month	the month
	RTC_year	the year
data	New contents.	

Return value:
The value requested by the user.

Example:
```
#include "4500drv.h"
void main(void)
{unsigned char sec=0,min=0,hour=12;
   adv_printf("Time    %02d:%02d:%02d\n",GetRTCtime(RTC_hour),
   GetRTCtime(RTC_min), GetRTCtime(RTC_sec));
   adv_printf("Set current time 12:00:00\n");
   SetRTCtime(RTC_sec,sec);
   SetRTCtime(RTC_min,min);
   SetRTCtime(RTC_hour,hour);
   adv_printf("Time    %02d:%02d:%02d\n",GetRTCtime(RTC_hour),
   GetRTCtime(RTC_min), GetRTCtime(RTC_sec));
}
```
Remarks:
None.

LED_init
LED_OFF
LED_ON

Syntax:
void LED_init(void)
void LED_OFF(int which_led)
void LED_ON(int which_led)

Description:
Turns LED lights on and off. The LED I/O port must be initialized first. It will take a little time for the light to stabilize following the signal for the turning on and turning off of the light.

Parameter:

Parameter	Value	Description
which_led	PWR	The PWR LED
	RUN	The RUN LED
	COMM	The COMM LED

Return value:
None.

Example:

```
#include "4500drv.h"
void main(void)
{
    LED_init();
    /* flash COMM led */
    while(1)
    {
        LED_ON(COMM);
        ADAMdelay(500);
        LED_OFF(COMM);
    }
}
```

Remarks:
None.

EraseSector
ProgramByte
ProgramSector

Syntax:
unsigned short EraseSector(unsigned long ulBase)
unsigned short ProgramByte(unsigned long ulAddress, BYTE byte)
unsigned short ProgramSector(unsigned long ulAddress_s, unsigned char far * SECTOR_DATA)

Description:
EraseSector: Erases a 64 KB sector of data in the 256 KB Flash memory
ProgramByte: Programs a byte of information into the 256 KB Flash memory. This feature supports data-logging or mass information storage.
ProgramSector: Programs an entire 32 KB sector of data of the global variable, SECTOR_DATA[], into 256 KB Flash memory.

Parameter	Description
ulBase	User-determined address range to be erased, taken from addresses in the range 0x80000L to 0xB0000L.
ulAddress	User–determined destination address for byte transfer, taken from the range 0x80000L to 0xBFFFFL.
byte	The data user wants to write into the specific byte in the Flash memory.
ulAddress_s	User-determined destination address in the Flash memory, taken from addresses in the range 0x80000L to 0xB8000L.
SECTOR_DATA	Pointer at the starting address in the origin memory of the user's data array.

Return value:
1 Successful transfer to Flash memory.
0 Error (destination already occupied, excess address range, or program error).

read_mem

Syntax:
unsigned char read_mem (int memory_segment , unsigned int i)

Description:
Reads far memory data, 256 KB Flash memory, from 0x80000L to
0xBFFFFL, where (the Absolute Address) = (SEG*16 + OFFSET). For example, (0x800FFL) = (0x8000*16 + 0x00FF).

Parameter	Description
memory_segment	User-determined address taken from the range 0x8000 to 0xBF00.
i	Offset for use in location of memory taken from the range 0x0000 to 0x0FFF.

Return value:
The value in memory storage at the indicated address.

Example:
```
#include "4500drv.h"
void main(void)
{
   unsigned char sector[32768];
   unsigned char data;
   unsigned long addr,sector_num;
   unsigned int i;

   adv_printf("erase sector 0x80000L\n");
   if(EraseSector(0x80000L))
   adv_printf("erase succeed \n");
   adv_printf("Write data(55) to 0x80000~0x80001\n");
   data=55;
   ProgramByte(0x80000L,data);
   ProgramByte(0x80000L+1,data);
   ProgramByte(0x80000L+2,data);
   for(i=0;i<3;i++)
   {
   adv_printf("read%d data=%d\n",i,read_mem(0x8000,0x0000+i));
   }
```

```
   adv_printf("erase sector 0x80000L\n");
   if(EraseSector(0x80000L))
   adv_printf("erase succeed \n");
   data = 1;
   for(i=0;i<32768;i++)
         *(sector+i)=data;
   adv_printf("Write data(0x01) to 0x80000~0x87FFF\n");
   ProgramSector(0x80000,&sector);
   for(i=0;i<100;i++)
   {
   adv_printf("read%d data=%d\n",i,read_mem(0x8000,0x0000+i));
   }
}
```

Remarks:
None.

Get_SysMem
Set_SysMem

Syntax:
unsigned char Get_SysMem(unsigned char which_byte)
void Set_SysMem(unsigned char which_byte, unsigned char data)

Description:
Get_SysMem: Reads a byte from security SRAM.
Set_SysMem: Writes a byte to security SRAM. Security SRAM supports 113 bytes for user storage of important information.

Parameter	Description
which_byte	From 0 to 112, user-determined.
data	Value to be saved.

Return value:
Get_SysMem: Get the value in a byte of security SRAM.
Set_SysMem: None.

Example:
```
#include "4500drv.h"
void main(void)
{
        unsigned char data[4] = {1,2,3,4};
        int i;
        /* save current value */
        for(i=10;i < 14;i++)
        {
                Set_SysMem(i, data[i-10]);
                adv_printf("data=%d\n",Get_SysMem(i));
        }
}
```

Remarks:
None

Get_NVRAM_Size
Set_NVRAM_Size

Syntax:
unsigned char Get_NVRAM_Size(void)
void Set_NVRAM_Size(unsigned char sector)

Description:
Get_NVRAM_Size: Gets the size of battery backup RAM.
Set_NVRAM_Size: Sets the size of battery backup RAM.
(The unit is sectors, each sector is 4KB in size. Maximum size is 384 KB theoretically.)

Parameter	Description
sector	NVRAM size in 4 KB sectors, from 1 to 96 sectors.

Return value:
Get_NVRAM_Size: sector Number of sectors NVRAM size is set to, from 1 to 96.
Set_NVRAM_Size: None.

Example:
```
#include "4500drv.h"
void main()
{
unsigned char sector;
sector = Get_NVRAM_Size();
adv_printf("Backup ram=%dKbyte\n",sector*4);
/*Set Bacup ram 40Kbyte*/
Set_NVRAM_Size(10);
}
```

Remarks:
None.

write_backup_ram
read_backup_ram

Syntax:
void write_backup_ram(unsigned long index, unsigned char data)
unsigned char read_backup_ram(unsigned long index)

Description:
write_backup_ram : Writes a byte to battery backup memory.
read_backup_ram : Reads the value in backup RAM at index address, maximum 384 KB total backup RAM, index = 0 – 393214

Parameter	Description
index	An index for data in the battery backup RAM, from 0 to 393214; maximum 384 KB battery backup SRAM in total.
data	A byte of data that the programmer wants to write to battery-protected SRAM.

Return value:
write_backup_ram: None.
read_backup_ram: The single-byte value in backup RAM at address index.

Example:
```
#include "4500drv.h"
void main()
{
        unsigned long addr;
        unsigned char data;
        /*write the data 0x55 into battery backup memory, index=10*/
        data=0x55;
        write_backup_ram(10,data);
        adv_printf("data=%x\n",read_backup_ram(10));
}
```

Remarks:
None

Timer_Init

Syntax:
int Timer_Init()

Description:
Initializes the timer built into the 80188 microprocessor. The return value "0" means the initialization of the time was successful. The
return value "1" means the timer had already been initialized.

Parameter
None.

Return value:
0: Initialization was successful.
1: The timer had already been initialized.

Remarks:
None.

Example:
Refer to *Release_All*

Timer_Reset

Syntax:
void Timer_Reset(int idx)

Description:
Reset the timer identified by the integer idx to its initial state.

Parameter	Description
idx	Timer index. (Timer ID)

Return value:
None.

Remarks:
None.

Example:
Refer to *Release_All*

Timer_Set

Syntax:
int Timer_Set(unsigned int msec)

Description:
Requests a timer from the microprocessor and then sets the time interval of the function. Timer intervals are set in 5 millisecond
increments. The function return value is an integer representing the ID of the timer function when it is successful.
A return value "-1" means the request failed. Programmers should consider whether an assigned timer has timed-out when programming for timer functions.
The value of the variable tmArriveCnt[idx] (where idx is the timer ID) can be checked to verify timer status. Value of 0 indicates that the timer is still counting. Values other than 0 mean the timer has timed- out.

Parameter	**Description**
msec	Time interval set, max. value is 65536.

Return value:
0~100 Function Succeed. Value represents the timer ID.
-1 Function failure.

Remarks:
Timer function calls in the ADAM-4500 Series are emulated as timer functions in a PLC. Applications using timer functions will run less
efficiently if the more timer functions are running simultaneously in a program.

Example:
Refer to *Release_All*

Release_All

Syntax:
void Release_All()

Description:
Releases all timer resources of the ADAM-4500 Series system.

Parameter
None.

Return value:
None.

Remarks:
None.

Example:

```
#include "4500drv.h"
void main()
{
        int idx;
        /*  Initializes the timer built into the 80188 microprocessor */
        Timer_Init();
        /*  Sets time interval of the timer to 1 second.        */
        idx=Timer_Set(1000);
        /*  Checks whether the timer has timed out          */
        while(tmArriveCnt[idx]==0)
        {
        /*  user can attend to other tasks...               */
                adv_printf("test");
        }

        /*  Resets the current timer to its initial state.      */
        Timer_Reset(idx);
        /*  Releases all timer resources                */
        Release_All();
}
```

WDT_clear
WDT_disable
WDT_enable

Syntax:
void WDT_clear(void)
void WDT_disable(void)
void WDT_enable(void)

Description:
WDT_clear: Clear watchdog timer.
WDT_disable: Disable watchdog timer.
WDT_enable: Enable watchdog timer.
When the watchdog timer is enabled, it will have to be cleared at least once every 1.6 seconds, or the system will automatically reboot. The watchdog timer default value is “disable”.

Parameter
None.

Return value:
None.

Example:
```
#include "4500drv.h"
void main(void)
{
        int i;
        WDT_enable();
        for(i=0;i<100;i++)
        {
                /*put your code in Here*/
                WDT_clear();
                /*put your code in Here*/
        }
        WDT_disable();
}
```

Remarks:
None

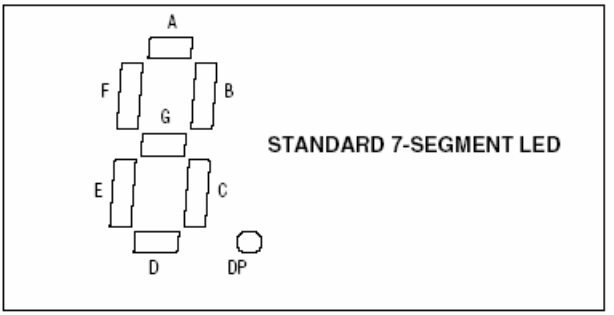
display_init
display_digit

Syntax:
void display_inti(unsigned char decode_mode)
int display_digit(unsigned char *data, unsigned char digit, unsigned char len)

Description:
7-segment display setting
Display_init: Initialize all 7-segment displays.
Display_digit: Writes the numeric into the all 7-segment displays.

Parameter	Description
decode_mode	Decode mode. (0 is No decode, 1 is Code B decode)
data	The display value.
digit	There are five 7-segment LED displays. This parameter determines which display is the starting LED display. It can be 1~5.
len	Total number of LED diplays you want to control.

⌘ **No Decode mode:**



	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Corresponding Segment Line	DP	A	B	C	D	E	F	G

The bit D0~D7 are the corresponding LED segment.
For example: data 30(hex) can show the “1” of the LED character. (D5 /D4 are 1 and others are 0)

⌘ Code B Decode mode:

7-SEGMENT CHARACTER	REGISTER DATA						ON SEGMENTS = 1							
	D7*	D6–D4	D3	D2	D1	D0	DP*	A	B	C	D	E	F	G
0		X	0	0	0	0		1	1	1	1	1	1	0
1		X	0	0	0	1		0	1	1	0	0	0	0
2		X	0	0	1	0		1	1	0	1	1	0	1
3		X	0	0	1	1		1	1	1	1	0	0	1
4		X	0	1	0	0		0	1	1	0	0	1	1
5		X	0	1	0	1		1	0	1	1	0	1	1
6		X	0	1	1	0		1	0	1	1	1	1	1
7		X	0	1	1	1		1	1	1	0	0	0	0
8		X	1	0	0	0		1	1	1	1	1	1	1
9		X	1	0	0	1		1	1	1	1	0	1	1
—		X	1	0	1	0		0	0	0	0	0	0	1
E		X	1	0	1	1		1	0	0	1	1	1	1
H		X	1	1	0	0		0	1	1	0	1	1	1
L		X	1	1	0	1		0	0	0	1	1	1	0
P		X	1	1	1	0		1	1	0	0	1	1	1
blank		X	1	1	1	1		0	0	0	0	0	0	0

*The decimal point is set by bit D7 = 1

Above this table, 7-SEGMENT CHARACTER is the value showed on LED displayed and REGISTER DATA is setting value. When the bit D7 is “1” (bit4~bit6 can be arbitrary bumber), the value will be a decimal point.

For example: data 3(hex) can show the number “**3**” of the LED character. And data 83 can show the number “**3.** ” of the LED character

Return value:
None.

Example:

Refer to C:\Program Files\Advantech\ADAM-4500 Series Utility \Source\Example\Basic_Function\ex1.C for this example

```
#include "4500drv.h"
#include "dos.h"
```

```
void      main(void)
{
  unsigned char data[5] = {1,2,3,4,5} ;
  int i;

          display_init(1);
          printf("please input 5 numbers (0 - 9):\n");
          for (i=0; i<5 ;i++)
          {
                    scanf("%d",&data[i]);
                    if (data[i]<0 || data[i] > 9)
                    {
                              printf("input error, exit");
                              exit(0);
                    }
          }

  display_digit(&data,1,5);
          exit(0);
}
```

Remarks:
None.

BatteryStatus

Syntax:
int BatteryStatus(void)

Description:
Check the power status of battery which is used by battery backup
SRAM.

Parameter
None.

Return value:

0	The power ststus of battery is almost run out. It is strongly recommend to change a new battery.
1	The power status of battery is still normal.

Example:
None.

Ver

Syntax:
void Ver(char *vstr)

Description:
Check Utility Library version.

Parameter	Description
vstr	Pointer to array of Utility Library version information

Return value:
None.

Example:

```
char library_ver[20];

void main(void)
{
        Ver(library_ver);

        adv_printf("The version of utility library is %s\n", library_ver);
}
```

5.3.2 I/O Module Functions (HIO*.LIB)

InitDIFilter

Syntax:
void InitDIFilter(int iCh, unsigned int MIN_Lo_Width, unsigned int
MIN_High_Width)

Description:
Set time interval of digital filter for DI channel.

Parameter	Description
iCh	Channel no. (0 ~ 4)
MIN_Lo_Width	Time interval of DI filter for Low state. (5 ~ 65535 msec)
MIN_High_Width	Time interval of DI filter for High state. (5 ~ 65535 msec)

Return Value:
None.

Remarks:
Reference Data:

Time Interval	Cut-off Frequency
15 ms	50 Hz
30 ms	20 Hz
50 ms	12 Hz

GetDIO (for DI only)
SetDO

Syntax:
UCHAR GetDIO(UCHAR i_ucModuleID, UCHAR i_ucMode, UCHAR
i_ucChannel, ULONG * o_ulValue)
UCHAR SetDO(UCHAR i_ucModuleID, UCHAR i_ucMode, UCHAR
i_ucChannel, ULONG i_ulValue)

Description:
Read/Write the value of digital input /ouput channels.

Parameter	Description
i_ucModuleID	Module ID.
i_ucMode	SingleChannel or AllChannels.
i_ucChannel	SingleChannel: the channel you use. AllChannels: don't care.
o_ulValue	DIO value read from module.
i_ulValue	DIO value wrritten to module.

Return Value:
If success, return 0. If fail, returns a negative number as follows:

Illegal_Setting	-5
Board_Not_Exist	-7

Remarks:
None

Ver_HIOLib

Syntax:
void Ver_HIOLib(char *vstr)

Description:
Get HIO Library version.

Parameter	Description
vstr	Pointer to array of HIO Library version information.

Return value:
None.

Example:
```
char library_ver[20];

void main(void)
{
        Ver_HIOLib(library_ver);

        adv_printf("The version of library is %s\n", library_ver);
}
```

Remarks:
None.

DIO Example:

Refer to C:\Program Files\Advantech\ADAM-4500 Series Utility \Source\Example\Basic_Function\Ex2.C for this example

```
#include "4500drv.h"
void main()
{
        int   tmpCnt=0;
        char c;
        unsigned char     type;
        unsigned long div,   dov;
        char Ver_Str[30];

        Ver_HIOLib(Ver_Str);
        printf("The HIO library version is %s\n", Ver_Str);

        /* ---- First scan for the existing IO modules -------*/
        type = Get_BoardID();

        /*----Show the module type of each slot on the screen ---*/
        if( type == EB50_ID)
                printf("IO slot is EB50\n");
        else
                printf("IO slot is Null\n");

        printf("press any key to continue...\n");getch();

        /*--- Digital I/O modules don't need to be initialized ---*/

        /*---- Forever loop until the user press any key */
        while(1)
        {
                tmpCnt++;
                /*--- Set DO Value     ------*/
                printf("\n\nSet and Get All Channels Test:\n");
                if((tmpCnt%2)==0)
                        dov=0;
                else
                        dov=0xf;
                if(SetDO(EB50_ID, AllChannels, 0, dov)==0)
                {
                        printf("Set DO value 0x%X, ", dov);
```

```
                    printf("press any key to continue..\n");getch();

                   if(GetDIO(EB50_ID, AllChannels, 0, &div)==0)
                    {
                            printf("DI value is 0x%X\n", div);
                    }
                    else
                            printf("Get DI failed\n");
            }
            else
                    printf("Set DO failed!\n");

            printf("\n\nSet and Get Sigle Channel Test:\n");
            if((tmpCnt%2)==0)
                    dov=0;
            else
                    dov=1;
            if(SetDO(EB50_ID, SingleChannel, 1, dov)==0)
            {
                    printf("Set DO channel 1 value %X, ", dov);
                    printf("press any key to continue..\n");getch();
                    if(GetDIO(EB50_ID, SingleChannel, 1,
                    &div)==0)
                    {
                            printf("Channel 1 DI value is %x\n",
                            div);
                    }
                    else
                            printf("Get DI failed\n");
            }
            else
                    printf("Set DO failed!\n");

            printf("press 'Q' to quit, the other key to continue..\n");
            c=getch();
            if( c == 'q' || c == 'Q')
                    break;
    }
}
```

5.3.3 Communication Functions (A4_COMM*.LIB)

SIO_Open

Syntax:
CHAR SIO_Open(UCHAR i_ucPort)

Description:
Initializes the COM port and interrupt service routine before other function calls use the COM port.

Parameter	**Description**
i_ucPort	The Port number you want to initial (see remarks).

Return value:
If success, return 0. If fail, returns a negative number as follows:

COM_already_installed	-1
Err_Access_COM	-2
No_Such_Port	-3

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_Close

Syntax:
CHAR SIO_Close(UCHAR i_ucPort)

Description:
Release resource of the specific COM port. If a user calls the SIO_Open function, the user must call this function to release the COM port before the user's program terminates.

Parameter	**Description**
i_ucPort	The Port number you want to release (see remarks).

Return value:
If success, return 0. If fail, returns a negative number as follows:
No_Such_Port -3

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_SetState

Syntax:
CHAR SIO_SetState(UCHAR i_ucPort, ULONG i_ulBaudRate, UCHAR i_ucParity, UCHAR i_ucDataBits, UCHAR i_ucStopBits)

Description:
Sets the parameters such as baud rate, parity, data bits and stop bits for the specific COM port.

Parameter	Description
i_ucPort	The Port number you want to use (see remarks).
i_ulBaudRate	The Baud Rate number you want to set up.
i_ucParity	The Parity you want to set up (see remarks).
i_ucDataBits	The data bits you want to set up (see remarks).
i_ucStopBits	The stop bits you want to set up (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows:

No_Such_Port	-3
COM_Not_Installed	-4
Illegal_Setting	-5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4
i_ucParity	NO_PARITY (0x00)	No parity
	ODD_PARITY (0x08)	Odd parity
	EVEN_PARITY (0x18)	Even Parity
i_ucDataBits	DATA5 (0x00)	5 Data Bits
	DATA6 (0x01)	6 Data Bits
	DATA7 (0x02)	7 Data Bits
	DATA8 (0x03)	8 Data Bits
i_ucStopBits	STOP1 (0x00)	1 Stop Bits
	STOP2 (0x04)	2 Stop Bits

SIO_RecvBytes

Syntax:
INT SIO_RecvBytes(UCHAR i_ucPort, UCHAR i_ucMode, UINT
i_uinBytes, UCHAR * o_ucDataBuf)

Description:
This function call is employed to received string data from the specific COM port. On success, this function call returns the total bytes has been read from the specific COM port. On fail, function call returns a negative value as error code.

Parameter	Description
i_ucPort	The Port number you want to use (see remarks).
i_ucMode	Decide to use Block mode or UnBlock mode. (When using Block mode, program will block here until i_uinBytes data is received. When using UnBlock mode, program will not block and return the total number of bytes data have been received directly.)
i_uinBytes	Number of bytes to be read.
o_ucDataBuf	A buffer for received data.

Return value:
If success, returns the total number of bytes read. If fail, return a negative number as follows:

No_Such_Port	-3
Illegal_Setting	-5
RequestOverQueueSize	-6

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4
i_ucMode	Block_Mode (0x01)	Block mode
	UnBlock_Mode (0x02)	UnBlock mode

SIO_SendBytes

Syntax:
INT SIO_SendBytes(UCHAR i_ucPort, UINT i_uinBytes, UCHAR *
i_ucDataBuf)

Description:
To send character(s) to a specified COM port.

Parameter	**Description**
i_ucPort	The Port number you want to send (see remarks).
i_uinBytes	Number of bytes to be sent.
i_ucDataBuf	A buffer for sending data.

Return value:
If success, return the number of bytes have been sent. If fail, return a negative number as follows
No_Such_Port -3

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_GetAvaiRecvBytes
SIO_GetAvaiSendBuf

Syntax:
INT SIO_GetAvaiRecvBytes(UCHAR i_ucPort)
INT SIO_GetAvaiSendBuf(UCHAR i_ucPort)

Description:
SIO_GetAvaiRecvBytes: Returns the number of bytes in input buffer.
SIO_GetAvaiSendBuf: Returns the number of bytes in output buffer.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).

Return value:
SIO_GetAvaiRecvBytes: If success, return the number of bytes data in input buffer. If fail, return a negative number as follows
No_Such_Port -3
SIO_GetAvaiSendBuf: If success, return the number of remaining available bytes space in output buffer. If fail, return a negative number as follows No_Such_Port -3

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_PurgeBuf

Syntax:
CHAR SIO_PurgeBuf(UCHAR i_ucPort, UCHAR i_ucFlag)

Description:
User can clear input buffer and output buffer together with argument i_ucFlag.

Parameter	Description
i_ucPort	The Port number you want to use (see remarks).
i_ucFlag	Decide which buffer you want to clear. You can choose to clear input buffer or output buffer (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows:
No_Such_Port -3
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4
i_ucFlag	Clear_RXBuffer (0x01)	Clear input buffer
	Clear_TXBuffer (0x02)	Clear output buffer

COM Port Communcation Example 1:

Refer to C:\Program Files\Advantech\ADAM-4500 Series Utility\Source\Example\Basic_Function\ExCOM.C for this example

```
#ifdef Product_5510
        #include "5510drv.h"
#elif defined Product_4500
        #include "4500drv.h"
#endif

#include <conio.h>
#include <dos.h>

#define DataLen50
#define RecvDataLen    30

void main()
{
        UCHAR tmpCOM, Idx=0;
        CHAR Data[DataLen];
        UCHAR Mode=0;          //0=>Receive, 1=>Send
        CHAR VerStr[30];
        UCHAR RecvMode;
        UCHAR nByte=0, nTotalByte=0;

        printf("Arthur New COM Port Library Test 3.3\n");
        printf("================================\n\n");

        Ver_COMLib(VerStr);
        printf("Com Library Version: %s\n", VerStr);

        printf("Enter receive mode( 1: Block mode, 0: unBlock mode):");
        scanf("%d", &tmpCOM);
        if(tmpCOM==0)
                RecvMode=UnBlock_Mode;
        else
                RecvMode=Block_Mode;

#ifdef Product_5510

        printf("Enter COM port selection( 1: COM1, 2: COM2, 3:
        COM4)");
```

```
        scanf("%d", &tmpCOM);
        if(tmpCOM==1)
                tmpCOM=COM1;
        else if(tmpCOM==2)
                tmpCOM=COM2;
        else if(tmpCOM==3)
                tmpCOM=COM4;

#elif defined Product_4500

        printf("Enter COM port selection( 1: COM1, 2: COM2, 3: COM3,
        4: COM4)");
        scanf("%d", &tmpCOM);
        if(tmpCOM==1)
                tmpCOM=COM1;
        else if(tmpCOM==2)
                tmpCOM=COM2;
        else if(tmpCOM==3)
                tmpCOM=COM3;
        else if(tmpCOM==4)
                tmpCOM=COM4;
#endif
        else
        {
                printf("Wrong selection!\n");
                return;
        }

        printf("Opening COM Port with 57600 baud...\n");

        if(SIO_Open(tmpCOM)!=0)
        {
                printf("error\n");
                return;
        }

        if(SIO_SetState(tmpCOM, (unsigned long)57600, NO_PARITY,
DATA8, STOP1)!=0)
        {
                printf("Set State Error\n");
                return;
        }
```

```
SIO_PurgeBuf(tmpCOM, Clear_RXBuffer);
SIO_PurgeBuf(tmpCOM, Clear_TXBuffer);

while(1)
{
        if(Mode==0)
        {
                if(RecvMode==Block_Mode)
                {

if(SIO_GetAvaiRecvBytes(tmpCOM)>=RecvDataLen)
                        {
                                if(SIO_RecvBytes(tmpCOM,
Block_Mode, RecvDataLen, Data)==RecvDataLen)
                                {
                                        if(Data[RecvDataLen-
                                        1]==0x0d)
                                        {
                                                Mode=1;
                                                Idx=0;
                                        }
                                        else
                                        {
                                                printf("Error
                                                Receive\n");
                                                return;
                                        }
                                }
                                else
                                {
                                        printf("Error
                                        Receive\n");
                                        return;
                                }
                        }
                        else
                        {
                                //do something else here
                        }
                }
```

```
                else if(RecvMode==UnBlock_Mode)
                {
                        while(nTotalByte<RecvDataLen)
                        {

if((nByte=SIO_RecvBytes(tmpCOM, UnBlock_Mode,
RecvDataLen, &Data[nTotalByte]))>=0)

                                {
                                        nTotalByte+=nByte;
                                }
                                else
                                {
                                        printf("Error
                                        Receive\n");
                                        return;
                                }
                        }

                        Mode=1;
                        Idx=0;
                        nTotalByte=0;
                }

        }
        else
        {

if(SIO_GetAvaiSendBuf(tmpCOM)>=RecvDataLen)
                {
                        if(SIO_SendBytes(tmpCOM,
                        RecvDataLen, Data)==RecvDataLen)
                        {
                                Idx=RecvDataLen;
                                if(Data[Idx-1]==0x0d)
                                {
                                        Mode=0;
                                        Idx=0;
                                }
                        }
```

```
                                        else
                                        {
                                                printf("Error Send\n");
                                                return;
                                        }
                                }
                                else
                                {
                                        printf("do something send\n");
                                        //do something else...
                                }
                        }
                }
        SIO_Close(tmpCOM);
}
```

SIO_MakeCheckSum

Syntax:
UINT SIO_MakeCheckSum(UCHAR * i_ucDataBuf, UINT i_uiLen)

Description:
Calculates the checksum of the string or data array (i_ucDataBuf).

Parameter	Description
i_ucDataBuf	The string or data array for which a user wants to calculate the checksum.
i_uiLen	The length of the string or the data array.

Return value:
The checksum value of the data array buffer

Remarks:
None.

SIO_MakeCRC16

Syntax:
UINT SIO_MakeCRC16(UCHAR * i_ucDataBuf, UINT i_uiLen)

Description:
Calculates the CRC 16-bit value of the string (i_ucDataBuf).

Parameter	Description
i_ucDataBuf	The string which you want to calculate CRC code.
i_uiLen	The length of string (i_ucDataBuf).

Return value:
The CRC16 code.

Remarks:
None.

SIO_Carrier

Syntax:
CHAR SIO_Carrier(UCHAR i_ucPort)

Description:
Detects the carrier signal of the specific COM port.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).

Return value:
If success, return TRUE(1). If fail, return a non-positive number as follows:

FALSE	0
Illegal_Setting	-5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_ClearBreak
SIO_SetBreak

Syntax:
CHAR SIO_ClearBreak(UCHAR i_ucPort)
CHAR SIO_SetBreak(UCHAR i_ucPort)

Description:
SIO_ClearBreak: Sets the specific COM port to clear BREAK signal.
SIO_SetBreak: Sets the specific COM port to send BREAK signal.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_GetLineStatus
SIO_SetLineParams
SIO_GetModemStatus

Syntax:
CHAR SIO_GetLineStatus(UCHAR i_ucPort)
CHAR SIO_SetLineParams(UCHAR i_ucPort, UCHAR i_ucParams)
CHAR SIO_GetModemStatus(UCHAR i_ucport)

Description:
SIO_GetLineStatus: Reads line control register of specific COM port.
SIO_SetLineParams: Writes to line control register of specific COM
port.
SIO_GetModemStatus: Reads modem status register of specific COM port.

Parameter	Description
i_ucPort	The Port number you want to use (see remarks).
i_ucParams	UART register parameter (refer to Appendix A for the 16C550 UART register document).

Return value:
If success, return the UART register value (refer to Appendix A for the 16C550 UART register document). If fail, return a negative number as follows:
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
	COM2	COM2
	COM3	COM3
	COM4	COM4

SIO_LowerRaise_RTS_DTR

Syntax:
CHAR SIO_LowerRaise_RTS_DTR(UCHAR i_ucPort, UCHAR i_ucL_R _Mode, UCHAR i_ucSignal)

Description:
Lower or raise the RTS/DTR signal.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).
i_ucL_R_Mode	Decide to raise or lower signal (see remarks).
i_ucSignal	Decide to set RTS or DTR signal (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows:
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1
i_ucL_R_Mode	RaiseSignal (0x01)	Raise signal
	LowerSignal (0x02)	Lower signal
i_ucSignal	Signal_RTS (0x01)	RTS signal
	Signal_DTR (0x02)	DTR signal

SIO_ModemInitial

Syntax:
CHAR SIO_ModemInitial(UCHAR i_ucPort)

Description:
Sets modem to initial status. Due to the ADAM-4500 Series system's construction, the modem can only be connected to COM1. This function resets the modem to the initial state. The command has the same effect as sending the ASCII command "atz" to the modem.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows:
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1

SIO_ModemAutoanswer

Syntax:
CHAR SIO_ModemAutoanswer(UCHAR i_ucPort)

Description:
Sets up modem to auto answer phone calls.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows:
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1

SIO_ModemCommand

Syntax:
CHAR SIO_ModemCommand(UCHAR i_ucPort, UCHAR * i_ucCmd
Str)

Description:
Sends an AT command string to the modem. For details, refer to the AT command document provided by the manufacturer.

Parameter	**Description**
i_ucPort	The Port number you want to use
i_ucCmdStr	Specifies command string, please refer to the AT command document

Return value:
If success, return 0. If fail, return a negative number as follows
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1

SIO_ModemCommand_State

Syntax:
CHAR SIO_ModemCommand_State(UCHAR i_ucPort)

Description:
Sets modem to command mode. In other words, this causes the modem to escape from data mode to command mode. The modem
will delay at least 3 seconds before switching back to command mode. This command has the same effect as sending the ASCII command "+++" to the modem.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows:
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1

SIO_ModemDial

Syntax:
CHAR SIO_ModemDial(UCHAR i_ucPort, UCHAR * i_ucTelenum)

Description:
Directs modem to connect to the specified telephone number.

Parameter	Description
i_ucPort	The Port number you want to use (see remarks).
i_ucTelenum	The phone number you would like modem to dial.

Return value:
If success, return 0. If fail, return a negative number as follows:
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1

SIO_ModemHandup

Syntax:
CHAR SIO_ModemHandup(UCHAR i_ucPort)

Description:
Sets the modem to hand up the telephone. The command has the same effect as sending the ASCII command “atho” to the modem.

Parameter	**Description**
i_ucPort	The Port number you want to use (see remarks).

Return value:
If success, return 0. If fail, return a negative number as follows:
Illegal_Setting -5

Remarks:

Parameter	Value	Description
i_ucPort	COM1	COM1

COM Port Communcation Example 2 (Modem):

Refer to C:\Program Files\Advantech\ADAM-4500 Series Utility\Source\Example\Basic_Function\Ex4.C for this example

```
#include "4500drv.h"

int get_modem_response(char *buf)
{
long i;
int index;
unsigned char c;

                        index=0;
                        for(i=0;i<10000; i++)
                        {
                          /*--- Get the receiving string from the COM1 port ---*/
                          if(SIO_RecvBytes(COM1, UnBlock_Mode, 1, &c)>0)
                          buf[index++]=c;

                          if(     index >0    && c == '\r')    /* end of command */
                          {
                            buf[index] =0;
                            adv_printf("Response : %s ",buf);
                            return(1);
                          }

                        }
                        return(0);
}


void main()
{
  char     c;
  int result_code;
  char
  buf[131];      int
  index;
  long     i;
  long     retry;

          SIO_ModemInitial(COM1);
```

```
while(1)
{
  adv_printf("---------- Main Menu ---------\n");
  adv_printf("0: Exit. \n");
  adv_printf("1: COM port setting. \n");
  adv_printf("2: Dial. \n");
  adv_printf("3: Set to auto-answer. \n");
  adv_printf("4: Set BREAK. \n");
  adv_printf("5: Hand up. \n");
  adv_printf("\n Please select a item to implement...\n");
  c=getch();
  switch(c)
  {
        case '0':
              return;

        case '1':
              /*--- Install the interrupt service routine for COM 1--*/
              if(SIO_Open(COM1)!=0)
              {
                    printf("error\n");
                    return;
              }
              if(SIO_SetState(COM1, (unsigned long)9600,
              NO_PARITY, DATA8, STOP1)!=0)
              {
                    printf("Set State Error\n");
                    return;
              }
              /*--- Show the data format on the screen ---*/
              adv_printf("COM port is COM1, baud rate is 9600 bps,
              data format is N,8,1\r\n");
              break;

        case '2':
              /*--- Send prefix ---*/
              SIO_ModemCommand(COM1, "AT");
              /*--- Wait about 1 second--*/
              retry=100000;
              for(i=0;i<retry;i++)
              {
                i++;i--;
              }
```

```
            /*--- Clear the buffers of the transmitter and receiver--*/
            SIO_PurgeBuf(COM1, Clear_RXBuffer);
            SIO_PurgeBuf(COM1, Clear_TXBuffer);
            /**--- Start to dial ---*/
            /*--- Set DTR is ON(1). --*/
            SIO_LowerRaise_RTS_DTR(COM1, RaiseSignal,
            Signal_DTR);
            /*--- Send the dialing command and phone number --*/
            sprintf(buf,"886222184867");
            SIO_ModemDial(COM1, buf);
            adv_printf("Command    : %s \n",buf);
            /*--- Wait for the response from the other end --*/
            if( get_modem_response(buf)== 1)
                adv_printf("Response : %s \n",buf);
            else
                adv_printf("Response : %s \n","No response");
            break;

        case '3':
            /*--- After one ring-bell, the phone is answered
            automatically.    */
            SIO_LowerRaise_RTS_DTR(COM1, RaiseSignal,
            Signal_DTR);
            SIO_ModemAutoanswer(COM1);
            adv_printf("Now is ready to get data...\n");
            break;

        case '4':
            /* Set Break */
            SIO_SetBreak(COM1);
            adv_printf("Now set a break to modem...\n\n\n");
            /*--- Wait about 0.3 second--*/
            retry=30000;  for(i=0;i<retry;i++)
            {
              i++;i--;
            }
            SIO_ClearBreak(COM1);
            adv_printf("Now clear the break ...\n\n\n");
            break;
```

```
        case '5':
                /*--- Set DTR line OFF --*/
                SIO_LowerRaise_RTS_DTR(COM1, LowerSignal,
                Signal_DTR);
                /*--- Wait about 0.3 second--*/
                retry=30000; for(i=0;i<retry;i++)
                {
                  i++;i--;
                }
                  /*---     Check whether DCD is off ---*/
                  /* if(!(com_get_modem_status(0x3F8)&0x80))
                  break;
                   */
                /*--- Go to modem command state ---*/
                SIO_ModemHandup(COM1);
                retry=3;
                do{
                   SIO_ModemHandup(COM1);
                   if(      get_modem_response(buf)== 1)
                   {
                     if( buf[0] ==0)
                           break;
                   }
                   adv_printf("retry %ld \n",4-retry);
                }while(--retry);
                adv_printf("Now is hand up...\n");
                break;
   }
  }
}
```

Ver_COMLib

Syntax:
void Ver_COMLib(char *vstr)

Description:
Get COM Port Library version.

Parameter	Description
vstr	Pointer to array of COM Port Library version information.

Return value:
None.

Example:
```
char library_ver[20];

void main(void)
{
        Ver_COMLib(library_ver);

        adv_printf("The version of library is %s\n", library_ver);
}
```

Remarks:
None.

5.3.4 MODBUS/RTU Functions (RTU*.LIB)

Before using Modbus Functions, please read the Modbus Quick Start below to understand how to use Modbus libraries :

Quick Start of MBTCP.Lib and MBRTU.Lib

Only two steps are needed to create a server or make a client query.

⌘ **How to Create a Modbus TCP/RTU Server?**

Modbus RTU Server:
Step 1: Use Modbus_COM_Init(...) function to initialize a COM port for Modbus RTU Server.
Step 2: Use ADAMRTU_ModServer_Create(...) to create Modbus RTU Server.

Modbus TCP Server:
Step 1: Use ADAMTCP_ModServer_Create(...) function to create a Modbus TCP server.
Step 2: Call ADAMTCP_ModServer_Update(...) function periodically to check if there is any client message and keep the server alive.

⌘ **How to Create a Modbus TCP/RTU Client?**

Modbus RTU Client:
Step 1: Use Modbus_COM_Init(...) function to initialize a COM port for Modbus RTU Client.
Step 2: Use ADAMModbusRTU_Read(...) to query data from server or use ADAMModbusRTU_Write(...) to write data to server.

Modbus TCP Client:
Step 1: Use ADAMTCP_Connect(...) to make a connection to server.
Step 2: Use ADAMModbusTCP_Read(...) to query data from server or use ADAMModbusTCP_Write(...) to write data to server.

⌘ **Example: (Modbus RTU Server and Modbus RTU client)**

Assume that we now need a Modbus RTU Server with valid address in the range of 40001-40008. Before starting to create a server, we need to define the slave ID for server

#define SlaveID 1

And second, we need to allocate a physical memory for address
40001-4008:
int ModbusAddr_Mem[8]; //ModbusAddr_Mem[0]=>40001,...,
//ModbusAddr_Mem[7]=>40008

Finally, we can create a server. The COM1 and Slave are used to specify that the COM1 is used as slave mode and dedicated for server use.
Modbus_COM_Init(COM1, Slave, (unsigned long)9600, NO_PARITY, DATA8, STOP1);
ADAMRTU_ModServer_Create(SlaveID, (unsigned char *)ModbusAddr_Mem,
sizeof(ModbusAddr_Mem));

But, how do we put the data to the address 40001-40008 for client queries? Just simply put any data you want to the physical memory: ***ModbusAddr_Mem[0]=0x1234; //40001=>0x1234***
ModbusAddr_Mem[7]=0x4321; //40008=>0x4321

OK, we've learned the techniques of creating a server. Now, let's see how to make query to get data at 40001-40008 from server. First, define some needed information for querying server.
#define Read_StartAddr 40001 //query data start from address 40001
#define Read_EndAddr 40008 //query data end to address 40008
#define SlaveID 1

Second, we need to allocate a physical memory for query data.
unsigned char Resp_From_Server[16];
//8 registers from 40001 to 40008.
//Therefore, we need at least a 16-byte physical memory.
int RespByteCount; //The total bytes of query data.

Finally, make a query. Unlike server programming, the mode is defined as Master instead of Slave.
Modbus_COM_Init(COM1, Master, (unsigned long)9600, NO_PARITY, DATA8, STOP1)
ADAMModbusRTU_Read(COM1, SlaveID, Read_StartAddr, Read_EndAddr, &RespByteCount, Resp_From_Server);

Now, the RespByteCount should report this query has get 16 bytes data. But, where is the data? The answer is quite simple as follows: ***Resp_From_Server[0] ==> 40001 Hi byte***
Resp_From_Server[1] ==> 40001 Lo byte

⌘ Example: (Modbus TCP Server and Modbus TCP client)

Assume that we now need a Modbus TCP Server with valid address in the range of 40001-40008. Before starting to create a server, we need to define some server information first such as the total numbers
of connections and the timeout setting.

```
#define TCP_Port 502      //502 is the standard port of modbus protocol
#define iTimeOut 3000     //3000 msec for timeout setting
#define iConns 20         //Only 20 client connections are avaliable
```

And second, we need to allocate a physical memory for address
40001-4008.

```
int ModbusAddr_Mem[8];      //ModbusAddr_Mem[0]=>40001,...,
                            //ModbusAddr_Mem[7]=>40008
```

Finally, we can create a server.

```
ADAMTCP_ModServer_Create(TCP_Port, iTimeOut, iConns, (unsigned
char*)ModbusAddr_Mem, sizeof(ModbusAddr_Mem));
while(1)
 //put ADAMTCP_ModServer_Update() inside infinite loop
{     //for calling ADAMTCP_ModServer_Update() periodically.
      if(ADAMTCP_ModServer_Update()==HasMessage)
           //check if there is any client message
      {
        ...
      }
}
```

But, how do we put the data to the address 40001-40008 for client
queries? Just simply put any data you want to the physical memory.

```
ModbusAddr_Mem[0]=0x1234;     //40001=>0x1234
ModbusAddr_Mem[7]=0x4321;     //40008=>0x4321
```

OK, we've learned the techniques of creating a server. Now, let's see how to make query to get data at 40001-40008 from server. First, define some needed information for querying server.

```
#define Server_Port 502     //502 is the standard port of modbus protocol
#define Server_IP "10.0.0.1"      //the IP of server
#define iTimeOut 4000             //4000 msec for timeout setting
#define Read_StartAddr 40001    //query data start from address 40001
#define Read_EndAddr 40008      //query data end to address 40008
#define SlaveID       1
```

Second, we need to allocate a physical memory for query data.
unsigned char Resp_From_Server[16];
//8 registers from 40001 to 40008. Therefore,
//we need at least a 16-byte physical memory.
int RespByteCount; //The total bytes of query data.

And, a descriptor for socket connection is also needed:
SOCKET SO;

Finally, make a query.
ADAMTCP_Connect(&SO, Server_IP, Server_Port);
ADAMModbusTCP_Read(&SO, iTimeout, SlaveID, Read_StartAddr, Read_EndAddr, &RespByteCount, Resp_From_Server);

Now, the RespByteCount should report this query has get 16 bytes data. But, where is the data? The answer is quite simple as follows: ***Resp_From_Server[0] ==> 40001 Hi byte***
Resp_From_Server[1] ==> 40001 Lo byte

Above is the conpect how to use Modbus protocol, and we also provide examples for reference, please refer the examples under **C:\Program Files\Advantech\ADAM-4500 Series Utility\Source\ Example\ModbusAppEx**

Note:
Modbus libraries come from Modbus standard protocol, and every function call can be mapped to a Modbus function. For example, ADAMRTU_ReadCoilStatus function is mapped to Modbus Function
code 01 (Read Coil Status). The following is the list of supported
Modbus function codes:

Modbus Function Codes	Related Modbus Libraries
01 Read Coil Status	**ADAMRTU_ReadCoilStatus** **ADAMTCP_ReadCoilStatus**
02 Read Input Status	**ADAMRTU_ReadInputStatus** **ADAMTCP_ReadInputStatus**
03 Read Holding Registers	**ADAMRTU_ReadHoldingRegs** **ADAMTCP_ReadHoldingRegs**
04 Read Input Registers	**ADAMRTU_ReadInputRegs** **ADAMTCP_ReadInputRegs**
05 Force Single Coil	**ADAMRTU_ForceSingleCoil** **ADAMTCP_ForceSingleCoil**
06 Preset Single Register	**ADAMRTU_PresetSingleReg** **ADAMTCP_PresetSingleReg**
15 Force Multiple Coils	**ADAMRTU_ForceMultiCoils** **ADAMTCP_ForceMultiCoils**
16 Preset Multiple Registers	**ADAMRTU_PresetMultiRegs** **ADAMTCP_PresetMultiRegs**

Moreover, we also provide four advanced function calls for beginners.

The Modbus function (01, 02, 03, 04) can be simply integrated into one “read” function as ADAMModbusRTU_Read (for Modbus RTU) or ADAMModbusTCP_Read (for Modbus TCP).

The Modbus function (05, 06, 15, 16) can be simply integrated into one “write” function as ADAMModbusRTU_Write (for Modbus RTU) or ADAMModbusTCP_Write (for Modbus TCP).

However, programmers should notice that those four functions are a little bit slower in performance than the original sixteen functions.

Modbus_COM_Init

Syntax:
int Modbus_COM_Init(int Port, int iMode, unsigned long iBaud, int iParity, int iFormat, int iStopBits)

Description:
Initial a COM port for Modbus/RTU connection.

Parameters	Value	Description
Port	COM1	Initial COM1
	COM2	Initial COM2
	COM3	Initial COM3
	COM4	Initial COM4
iMode	Slave	Modbus/RTU slave mode
	Master	Modbus/RTU master mode
iBaud	9600, etc …	The value of baud rate
iparity	NO_PARITY	No parity
	ODD_PARITY	Odd parity
	EVEN_PARITY	Even parity
	ONE_PARITY	Parity=1
	ZERO_PARITY	Parity=0
iFormat	DATA5	5 data bit
	DATA6	6 data bit
	DATA7	7 data bit
	DATA8	8 data bit
iStopBits	STOP1	One stop bit
	STOP2	Two stop bits

Return value:
0 No error occurs.
1 COM_already_installed: COM port has been installed before.
2 Err_Access_COM: Error occurs when try to access COM port.

Example:

```
if(Modbus_COM_Init(COM2, Master, (unsigned long)9600,
NO_PARITY, DATA8, STOP1)!=0)
        {
                adv_printf("error\n");
                return;
        }

        adv_printf("init success!!\n");
```

Modbus_COM_Release

Syntax:
void Modbus_COM_Release(int Port)

Description:
Release the COM port of Modbus connection.

Parameters	Value	Description
Port	1	COM1
	2	COM2
	3	COM3
	4	COM4

Return value:
None.

Error_Code

Syntax:
int Error_Code(void)

Description:
When following function call gets error return, this function can get the exact error code for user:
ADAMRTU_ForceMultiCoils()
ADAMRTU_ForceSingleCoil()
ADAMRTU_PresetMultiRegs()
ADAMRTU_PresetSingleReg()
ADAMRTU_ReadCoilStatus()
ADAMRTU_ReadHoldingRegs()
ADAMRTU_ReadInputRegs()
ADAMRTU_ReadInputStatus()

Parameters
None.

Return value:

NULL	No exception error returned.
Erro Code	Exception error returned.

Error code:

91	Invalid Response
92	COM Port Initial or Mode Error
93	COM Port Time Out

Example:
Refer to *ADAMRTU_ForceMultiCoils*

ADAMRTU_ForceMultiCoils

Syntax:
bool ADAMRTU_ForceMultiCoils(int iPort, int Slave_Addr, int CoilIndex, int TotalPoint, int TotalByte, unsigned char szData[])

Description:
"0F HEX" command of Modbus/RTU function code

Parameters	**Description**
iPort	COM port number
Slave_Addr	Slave address
CoilIndex	Coil address
TotalPoint	Quantity of coils
TotalByte	Byte count
szData[]	Force Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
HostData[0]=0xf0;

if(!ADAMRTU_ForceMultiCoils(COM1, 0x02, 0x64, 0x08, 0x01,
HostData))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
        }
        else
                adv_printf("Success!!");
```

ADAMRTU_ForceSingleCoil

Syntax:
bool ADAMRTU_ForceSingleCoil(int iPort, int i_iAddr, int i_iCoilIndex, int i_iData)

Description:
"05 HEX" command of Modbus/RTU function code.

Parameters	**Description**
iPort	COM port number
i_iAddr	Slave address
i_iCoilIndex	Coil address
int i_iData	Force Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
if(!ADAMRTU_ForceSingleCoil(COM1, 0x02, 0x65, 0))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
        }
        else
                adv_printf("Success!!");
```

ADAMRTU_PresetMultiRegs

Syntax:
bool ADAMRTU_PresetMultiRegs(int iPort, int i_iAddr, int i_iStartReg, int i_iTotalReg, int i_iTotalByte, unsigned char i_szData[])

Description:
"10 HEX" command of Modbus RTU function code

Parameters	**Description**
iPort	COM port number
i_iAddr	Slave address
i_iStartReg	Starting Address
i_iTotalReg	No. of Registers Hi
i_iTotalByte	Byte Count
i_szData[]	Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
HostData[0]=0x12;
HostData[1]=0x56;
HostData[2]=0x38;
HostData[3]=0x09;

if(!ADAMRTU_PresetMultiRegs(COM1, 0x02, 0x64, 2, 4, HostData))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
                return;
        }
        else
                adv_printf("Success!!");
```

ADAMRTU_PresetSingleReg

Syntax:
bool ADAMRTU_PresetSingleReg(int iPort, int i_iAddr, int i_iRegIndex, int i_iData)

Description:
"06 HEX" command of Modbus RTU function code

Parameters	**Description**
iPort	COM port number
i_iAddr	Slave Address
i_iRegIndex	Register Address
i_iData	Preset Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
if(!ADAMRTU_PresetSingleReg(COM1, 0x02, 0x68, 0x1234))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
                return;
        }
        else
                adv_printf("Success!!");
```

ADAMModbusRTU_Write

Syntax:
bool ADAMModbusTCP_Write(SOCKET * SO, int WaitMilliSec, int Slave_Addr, unsigned long i_iStartAddr, unsigned long i_iEndAddr, unsigned char i_szData[]);

Description:
Presets values or forces coils into a sequence of holding registers (4X references) or a sequence of coilds(0X references).

Parameters	Description
iPort	COM port number
Slave_Addr	Address ID of slave device (valid slave device addresses id are in the range of 0-247 decimal)
i_iStartAddr	The start address of slave device
i_iEndAddr	The end address of slave device
i_szData	The buffer of query data contents

Return value:
ADAMModbusRTU_Write() returns TRUE if it is successful. On an error, a value of FALSE will be return and the function Error_Code() can be used to get the last error.

Example:

please refer the examples under
C:\Program Files\Advantech\ADAM-4500 Series Utility\Source\ Example\ModbusAppEx

ADAMRTU_ReadCoilStatus

Syntax:
bool ADAMRTU_ReadCoilStatus(int iPort, int i_iAddr, int i_iStartIndex, int i_iTotalPoint, int *o_iTotalByte, unsigned char o_szData[])

Description:
"01HEX" command of Modbus RTU function code.

Parameters	**Description**
iPort	COM port number
i_iAddr	Slave Address
i_iStartIndex	Starting Address
i_iTotalPoint	No. of Points
o_iTotalByte	Byte Count
o_szData[]	Coil Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
if(!ADAMRTU_ReadCoilStatus(COM1, 0x02, 0x6E, 0x01,
   &DataByteCount, HostData))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
        }
        else
        {
                adv_printf("Status: ");
                for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
                {
                        adv_printf("%02X", HostData[tmpcnt]);
                }
                adv_printf("\n");
        }
```

ADAMRTU_ReadHoldingRegs

Syntax:
bool ADAMRTU_ReadHoldingRegs(int iPort, int i_iAddr, int i_iStartIndex, int i_iTotalPoint, int *o_iTotalByte, unsigned char o_szData[])

Description:
"03 HEX" command of Modbus RTU function code.

Parameters	Description
iPort	COM port number
i_iAddr	Slave Address
i_iStartIndex	Starting Address
i_iTotalPoint	No. of Points
o_iTotalByte	Byte Count
o_szData[]	Register Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
if(!ADAMRTU_ReadHoldingRegs(COM1, 0x02, 0x65, 0x01,
    &DataByteCount, HostData))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
        }
        else
        {
                adv_printf("Status: ");
                for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
                {
                        adv_printf("%02X", HostData[tmpcnt]);
                }
                adv_printf("\n");
        }
```

ADAMRTU_ReadInputRegs

Syntax:
bool ADAMRTU_ReadInputRegs(int iPort, int i_iAddr, int i_iStartIndex, int i_iTotalPoint, int *o_iTotalByte, unsigned char o_szData[])

Description:
"04 HEX" command of Modbus RTU function code.

Parameters	**Description**
iPort	COM port number
i_iAddr	Slave Address
i_iStartIndex	Starting Address
i_iTotalPoint	No. of Points
o_iTotalByte	Byte Count
o_szData[]	Register Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
if(!ADAMRTU_ReadInputRegs(COM1, 0x02, 0x65, 0x01,
    &DataByteCount, HostData))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
        }
        else
        {
                adv_printf("Status: ");
                for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
                {
                        adv_printf("%02X", HostData[tmpcnt]);
                }
                adv_printf("\n");
```

ADAMRTU_ReadInputStatus

Syntax:
bool ADAMRTU_ReadInputStatus(int iPort, int i_iAddr, int i_iStartIndex, int i_iTotalPoint, int *o_iTotalByte, unsigned char o_szData[])

Description:
"02 HEX" command of Modbus RTU function code.

Parameters	**Description**
iPort	COM port number
i_iAddr	Slave Address
i_iStartIndex	Starting Address
i_iTotalPoint	No. of Points
o_iTotalByte	Byte Count
o_szData[]	Inputs Data

Return value:

TRUE	No error occurs.
FALSE	Error occurs, call *Error_Code()* for exact error codes.

Example:

```
if(!ADAMRTU_ReadInputStatus(COM1, 0x02, 0x64, 0x08,
    &DataByteCount, HostData))
        {
                adv_printf("err code is %d\n", Error_Code());
                adv_printf("fail send..");
        }
        else
        {
                adv_printf("Status: ");
                for(tmpcnt=0; tmpcnt<DataByteCount; tmpcnt++)
                {
                        adv_printf("%02X", HostData[tmpcnt]);
                }
                adv_printf("\n");
        }
```

ADAMModbusRTU_Read

Syntax:
bool ADAMModbusRTU_Read(int iPort, int Slave_Addr, unsigned long i_iStartAddr, unsigned long i_iEndAddr, unsigned int *o_iByteOfResp, unsigned char o_szResp[]);

Description:
Reads the On/OFF status of 0X/1X references(coils) or the binary contents of 3X/4X references(registers) in the slave.

Parameters	Description
iPort	COM port number
Slave_Addr	Address ID of slave device (valid slave device addresses id are in the range of 0-247 decimal)
i_iStartAddr	The start address of slave device
i_iEndAddr	The end address of slave device
i_iTotalPoint	The total number of coils/regs to be read
o_iByteOfResp	The total bytes of the response data contents
o_szResp	The buffer of response data contents

Return value:
ADAMModbusRTU_Read() returns TRUE if it is successful. On an error, a value of FALSE will be return and the function Error_Code() can be used to get the last error.

Example:

please refer the examples under
C:\Program Files\Advantech\ADAM-4500 Series Utility\Source\ Example\ModbusAppEx.

ADAMRTU_ModServer_Create

Syntax:
void ADAMRTU_ModServer_Create(int slave_addr, unsigned char *
ptr_mem, unsigned int size_of_mem)

Description:
Create Modbus/RTU Server function.

Parameters	Description
slave_addr	Slave address of Modbus/RTU Server
ptr_mem	Share memory
size_of_mem	Size of share memory

Return value:
None.

Example:

```
ADAMRTU_ModServer_Create(3, (unsigned char *)Share_Mem,
sizeof(Share_Mem));

        adv_printf("server started..\n");

        while(1)
        {
                if(predate != Share_Mem[0])
                {
                adv_printf("40001 is %X\n", Share_Mem[0]);
                  //strongly recommend use adv_printf() instead of
                  printf()
                predate = Share_Mem[0];
                }
        }
```

Ver_RTU_Mod

Syntax:
void Ver_RTU_Mod(char *vstr)

Description:
Check Modbus/RTU Library version.

Parameter	Description
vstr	Pointer to array of Modbus/RTU Library version information

Return value:
None.

Example:

```
char library_ver[20];

void main(void)
{
        Ver_RTU_Mod(library_ver);

        adv_printf("The version of library is %s\n", library_ver);
}
```

5.3.5 MODBUS/TCP Functions (MBTCP*.LIB)

Ver_TCP_Mod

Syntax:
void Ver_TCP_Mod(char *vstr)

Description:
Check Modbus/TCP Library version.

Parameter	Description
vstr	Pointer to array of Modbus/TCP Library version information.

Return value:
None.

Example:

```
char library_ver[20];

void main(void)
{
        Ver_TCP_Mod(library_ver);
        adv_printf("The version of library is %s\n", library_ver);
}
```

Modbus TCP Client Functions:

ReturnErr_code

Syntax:
int ReturnErr_code(void)

Description:
When following function call gets error return, this function can get the exact error code for user.

ADAMTCP_ForceMultiCoils()
ADAMTCP_ForceSingleCoil()
ADAMTCP_PresetMultiRegs()
ADAMTCP_PresetSingleReg()
ADAMTCP_ReadCoilStatus()
ADAMTCP_ReadHoldingRegs()
ADAMTCP_ReadInputRegs()
ADAMTCP_ReadInputStatus()

Parameters
None.

Return value:

NULL	No error occurs.
Erro Code	Exception error returned.

Error code:

01	ILLEGAL FUNCTION
02	ILLEGAL DATA ADDRESS
03	ILLEGAL DATA VALUE
04	SLAVE DEVICE FAILURE
05	ACKNOWLEDGE
06	SLAVE DEVICE BUSY
07	NEGATIVE ACKNOWLEDGE
08	MEMORY PARITY ERROR

Example:

```
SOCKET SO_4500;
…

if(ADAMTCP_ReadCoilStatus(&SO_4500, 50, 0x01, 0x11, 0x10,
      &DataByteCount, HostData)<=0)
   {
         perror("ADAMTCP_ReadCoilStatus()\n");
         adv_printf("err code is %d\n", ReturnErr_code());
         ADAMTCP_Disconnect(&SO_4500);
         return 0;
   }
```

ADAMTCP_Connect

Syntax:
int ADAMTCP_Connect(SOCKET * SO, char * Target_IP, int Target_Port)

Decription:
Connect to Modbus/TCP Server.

Parameters	Description
SO	A descriptor identifying an unconnected socket.
Target_IP	Modbus/TCP server IP.
Target_Port	Server port for the connection.

Return value:

TRUE	No error occurs
-1	Error occurs when gets the host name
-2	The socket is invalid when initializes the socket
-3	Error occurs when connects to Modbus/TCP server

Example:

```
if(ADAMTCP_Connect(&SO_4500, ServerIP, Server_Port)<=0)
   {
      perror("ADAMTCP_Connect()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
   }
```

ADAMTCP_Disconnect

Syntax:
bool ADAMTCP_Disconnect(SOCKET * SO)

Description:
Disconnect to Modbus/TCP Server.

Parameters

SO	A descriptor identifying the connected socket to Modbus/TCP server.

Return value:

TRUE	No error occurs
FALSE	There is error occurs

Example:

```
if(ADAMTCP_Connect(&SO_4500, ServerIP, Server_Port)<=0)
  {
    perror("ADAMTCP_Connect()\n");
    ADAMTCP_Disconnect(&SO_4500);
    return 0;
  }
```

ADAMTCP_ForceMultiCoils

Syntax:
int ADAMTCP_ForceMultiCoils(SOCKET * SO, int WaitMilliSec, int
Slave_Addr, int CoilIndex, int TotalPoint, int TotalByte, unsigned char szData[])

Description:
"0F HEX" command of Modbus TCP function code.

Parameters	**Descerip tion**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration (msec unit) for the response from Modbus/TCP server
Slave_Addr	Slave address
CoilIndex	Coil address
TotalPoint	Quantity of coils
TotalByte	Byte count
szData[]	Force Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:

```
HostData[1]=~0x33;
//force channel status to 0x3333
if(ADAMTCP_ForceMultiCoils(&SO_4500, 50, 0x01, 0x21, 0x10, 0x02,
      HostData)<=0)
   {
      perror("ADAMTCP_ForceMultiCoils()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
   }
```

ADAMTCP_ForceSingleCoil

Syntax:
int ADAMTCP_ForceSingleCoil(SOCKET * SO, int WaitMilliSec, int
Slave_Addr, int CoilIndex, int Data)

Description:
"05 HEX" command of Modbus TCP function code.

Parameters	**Descerip tion**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration (msec unit) for the response from Modbus/TCP server
Slave_Addr	Slave address
CoilIndex	Coil address
Data	Force Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:

```
if(ADAMTCP_ForceSingleCoil(&SO_4500, 50, 0x01, 0x25, 1)<=0)
   {
      perror("ADAMTCP_ForceSingleCoil()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
   }
```

ADAMTCP_PresetMultiRegs

Syntax:
int ADAMTCP_PresetMultiRegs(SOCKET * SO, int WaitMilliSec, int Slave_Addr, int StartReg, int TotalReg, int TotalByte, unsigned char
Data[])

Description:
"10 HEX" command of Modbus TCP function code.

Parameters	**Descerіption**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration (msec unit) for the response from Modbus/TCP server
Slave_Addr	Slave address
StartReg	Starting address
TotalReg	No. of registers
TotalByte	Byte count
szData[]	Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:

```
HostData[0]=0x07;
HostData[1]=0x00;
HostData[2]=0x07;
HostData[3]=0x00;

if(ADAMTCP_PresetMultiRegs(&SO_4500, 50, 0x01, 0x19, 0x02, 4,
      HostData)<=0)
   {
      perror("ADAMTCP_PresetMultiRegs()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
```

```
}
```

ADAMTCP_PresetSingleReg

Syntax:
int ADAMTCP_PresetSingleReg(SOCKET * SO, int WaitMilliSec, int
Slave_Addr, int RegIndex, int Data)

Description:
"06 HEX" command Modbus TCP function code.

Parameters	**Descerip tion**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration (msec unit) for the response from Modbus/TCP server
Slave_Addr	Slave address
RegIndex	Register address
Data	Preset Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:

```
if(ADAMTCP_PresetSingleReg(&SO_4500, 50, 0x01, 0x19,
0x07ff)<=0)
   {
      perror("ADAMTCP_PresetSingleReg()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
   }
```

ADAMModbusTCP_Write

Syntax:
bool ADAMModbusTCP_Write(SOCKET * SO, int WaitMilliSec, int
Slave_Addr, unsigned long i_iStartAddr, unsigned long
i_iEndAddr, unsigned char i_szData[]);

Description:
Presets values or forces coils into a sequence of holding registers (4X references) or a sequence of coilds(0X references).

Parameters	**Description**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	The maximum time of ADAMModbusTCP_Write()
Slave_Addr	Address ID of slave device (valid slave device addresses id are in the range of 0-247 decimal)
i_iStartAddr	The start address of slave device
i_iEndAddr	The end address of slave device
i_szData	The buffer of query data contents

Return value:
ADAMModbusTCP_Write() returns TRUE if it is successful. On an error, a value of FALSE will be return and the function Error_Code()
can be used to get the last error.

Example:

please refer the examples under
C:\Program Files\Advantech\ADAM-4500 Series Utility\Source\ Example\ModbusAppEx

ADAMTCP_ReadCoilStatus

Syntax:
int ADAMTCP_ReadCoilStatus(SOCKET * SO, int WaitMilliSec, int
Slave_Addr, int StartIndex, int TotalPoint, int * ByteCount, char *
wData)

Description:
"01 HEX" command of Modbus TCP function code

Parameter	**Descerip tion**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration (msec) for the response from Modbus/TCP server
Slave_Addr	Slave address
StartIndex	Starting address
TotalPoint	No. of points
ByteCount	Byte count
wData	Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:

```
if(ADAMTCP_ReadCoilStatus(&SO_4500, 50, 0x01, 0x11, 0x10,
   &DataByteCount, HostData)<=0)
   {
      perror("ADAMTCP_ReadCoilStatus()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
   }
   else
   {    adv_printf("Adam-4500 Status: ");
      for(tmp=0; tmp<DataByteCount; tmp++)
      {
         adv_printf("%2X", HostData[tmp]);
      }
```

```
        adv_printf("\n");
    }
```

ADAMTCP_ReadHoldingRegs

Syntax:
int ADAMTCP_ReadHoldingRegs(SOCKET * SO, int WaitMilliSec, int Slave_Addr, int StartIndex, int TotalPoint, int * ByteCount, char *
wData)

Description:
"03 HEX" command of Modbus TCP function code.

Parameters	**Descerip tion**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration(msec unit) for the response from Modbus/TCP server
Slave_Addr	Slave address
StartIndex	Starting address
TotalPoint	No. of points
ByteCount	Byte count
wData	Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:
//This example demonstrates how to use ADAMTCP_ ReadHoldingRegs() to query remote Modbus TCP module, such as
ADAM-5000/TCP. And it queries all channels of ADAM-5024 inserted in Slot 3 on the ADAM-5000/TCP.

```
if((errno=ADAMTCP_ReadHoldingRegs(&SO_4500,  50,  0x01,
0x19,
   0x08, &DataByteCount, HostData))<=0)
   {
      perror("ADAMTCP_ReadHoldingRegs()\n");
      adv_printf("errno is %d\n", errno);
```

```
    ADAMTCP_Disconnect(&SO_4500);
    return 0;
}
```

```
else
{
    adv_printf("Adam-5024 Status: ");
    for(tmp=0; tmp<DataByteCount; tmp++)
    {
        adv_printf("%02X", HostData[tmp]);
    }
    adv_printf("\n");
}
```

ADAMTCP_ReadInputRegs

Syntax:
int ADAMTCP_ReadInputRegs(SOCKET * SO, int WaitMilliSec, int
Slave_Addr, int StartIndex, int TotalPoint, int * ByteCount, char *
wData)

Description:
"04 HEX" command of Modbus TCP function code.

Parameter	**Descerip tion**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration(msec unit) for the response from Modbus/TCP server
Slave_Addr	Slave address
StartIndex	Starting address
TotalPoint	No. of points
ByteCount	Byte count
wData	Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:
//This example demonstrates how to use ADAMTCP_ReadInputRegs to query remote Modbus TCP module, such as ADAM-5000/TCP. And here it queries all chanels of ADAM-5024 inserted in Slot 3 on the ADAM-5000/TCP.

```
if((errno=ADAMTCP_ReadInputRegs(&SO_4500, 50, 0x01, 0x19,
   0x08, &DataByteCount, HostData))<=0)
   {
      perror("ADAMTCP_ReadInputRegs()\n");
      adv_printf("errno is %d\n", errno);
```

```
    ADAMTCP_Disconnect(&SO_4500);
    return 0;
}
```

```
else
{
    adv_printf("Adam-5024 Status: ");
    for(tmp=0; tmp<DataByteCount; tmp++)
    {
        adv_printf("%02X", HostData[tmp]);
    }
    adv_printf("\n");
}
```

ADAMTCP_ReadInputStatus

Syntax:
int ADAMTCP_ReadInputStatus(SOCKET * SO, int WaitMilliSec, int
Slave_Addr, int StartIndex, int TotalPoint, int * ByteCount, char *
wData)

Description:
"02 HEX" command Modbus TCP function code.

Parameters	**Descerip tion**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	Set duration (msec unit) for the response from Modbus/TCP server
Slave_Addr	Slave address
StartIndex	Starting address
TotalPoint	No. of points
ByteCount	Byte count
wData	Data

Return value:

TRUE	No error occurs
0	Time out error when receive modbus query message from Modbus/TCP server
-1	Error occurs when send modbus query message to Modbus/TCP server
-2	Error occurs when receive modbus query message from Modbus/TCP server

Example:
//This example demonstrates how to use ADAMTCP_ReadInputStatus to query remote Modbus TCP module, such as ADAM-5000/TCP. And here it queries ADAM-5051 inserted in Slot 1 on the ADAM-5000/TCP.

```
if(ADAMTCP_ReadInputStatus(&SO_4500, 50, 0x01, 0x11, 0x10,
   &DataByteCount, HostData)<=0)
   {
      perror("ADAMTCP_ReadInputStatus()\n");
      ADAMTCP_Disconnect(&SO_4500);
      return 0;
   }
```

```
else
   {
      adv_printf("Adam-5051 Status: ");
      for(tmp=0; tmp<DataByteCount; tmp++)
         {
            adv_printf("%2X", HostData[tmp]);
         }
      adv_printf("\n");
   }
```

ADAMModbusTCP_Read

Syntax:
bool ADAMModbusTCP_Read(SOCKET * SO, int WaitMilliSec, int Slave_Addr, unsigned long i_iStartAddr, unsigned long i_iEndAddr, unsigned int *o_iByteOfResp, unsigned char o_szResp[]);

Description:
Reads the On/OFF status of 0X/1X references(coils) or the binary contents of 3X/4X references(registers) in the slave.

Parameters	**Description**
SO	The socket connected to Modbus/TCP server
WaitMilliSec	The maximum time of ADAMModbusTCP_Read()
Slave_Addr	Address ID of slave device (valid slave device addresses id are in the range of 0-247 decimal)
i_iStartAddr	The start address of slave device
i_iEndAddr	The end address of slave device
i_iTotalPoint	The total number of coils/regs to be read
o_iByteOfResp	The total bytes of the response data contents
o_szResp	The buffer of response data contents

Return value:
ADAMModbusRTU_Read() returns TRUE if it is successful. On an error, a value of FALSE will be return and the function Error_Code() can be used to get the last error.

Example:

please refer the examples under
C:\Program Files\Advantech\ADAM-4500 Series Utility\Source\ Example\ModbusAppEx.

Modbus TCP Server Functions:

ADAMTCP_ModServer_Create

Syntax:
int ADAMTCP_ModServer_Create(int Host_Port, unsigned long waittimeout, unsigned int numberConns, unsigned char * ptr_mem, int size_mem)

Description:
Create a Modbus/TCP Server.

Parameters	**Description**
Host_Port	The port for Modbus/TCP server
Waittimeout	Time out value, 0~0xffffffff milli-second
NumberConns	Maximum connections for client
ptr_mem	Share memory
size_mem	The size of share memory

Return value:

0	No error occurs
91	Invalid socket
92	Error occurs when associates a local socket address with a socket
93.	Error occurs when sets up the socket mode
94	Error occurs when listens to the incoming socket

Example:

```
if((err_code=ADAMTCP_ModServer_Create(502, 5000, 20,
  (unsigned char *)Share_Mem, sizeof(Share_Mem)))!=0)
        {
                adv_printf("error code is %d/n", err_code);
        }
adv_printf("Server started, wait for connect...\n");
```

ADAMTCP_ModServer_Update

Syntax:
int ADAMTCP_ModServer_Update(void)

Description:
Update the Modbus/TCP Server. The Modbus/TCP server needs to be updated by calling ADAMTCP_ModServer_Update() function continuously to keep server alive.

Parameters
None.

Return value:
1 New message has come in
0 No new message comes in

Example:

```
while(1)
{
        iState = ADAMTCP_ModServer_Update();        //second step
        if(iState) //if has message, show the data at address 40001
        {
                if(pre_data != Share_Mem[0])
                {
                adv_printf("40001 is %X\n", Share_Mem[0]);
                  //notice: printf() will decrease server performance
                pre_data = Share_Mem[0];
                }
        }
}
```

ADAMTCP_ModServer_Release

Syntax:
void ADAMTCP_ModServer_Release(void)

Description:
Release Modbus/TCP Server.

Parameters
None.

Returned value:
None.

5.4.6 Socket Functions (SOCKET*.LIB)

TCP/IP SOCKETS API Overview

This section describes the SOCKETS API, which is compatible with the BSD Sockets API and also the Winsock API. The definitions and prototypes for the C environment are supplied in SOCKET.H, while the implementation of the C interface is in SOCKET.C. The SOCKETS API is implemented as a layer on top of the Compatible API (CAPI) and provides an interface to the socket and name resolution facilities provided by the Datalight DOS SOCKETS product. It also provides the database functions of BSD Sockets and Winsock.

A *socket* is an end-point for a connection and is defined by the combination of a host address (also known as an IP address), a port number (or communicating process ID), and a transport protocol, such as UDP or TCP. Two connected SOCKETS using the same transport protocol define a connection. The API uses a socket handle, sometimes referred to as simply a socket. The socket handle is required by most function calls in order to access a connection. The socket handle used is the same as a normal socket as used in CAPI.
A socket handle is obtained by calling the **socket**() function. A socket handle can only be used for a single connection. When no longer required, such as when a connection has been closed, the socket
handle must be released by calling **closesocket**(). Socket handles
are positive numbers greater than 63.

Types of Service

SOCKETS can be used with one of two service types:
SOCK_STREAM (using TCP).
SOCK_DGRAM (using UDP).

A stream connection provides for the bi-directional, reliable, sequenced, and unduplicated flow of data without record boundaries. No broadcast facilities can be used with a stream connection.
A datagram connection supports bi-directional flow of data that is not guaranteed to be sequenced, reliable, or unduplicated. That is, a process receiving messages on a datagram socket may find messages duplicated, and,

possibly, in an order different from the order in which it was sent. An important characteristic of a datagram connection is that record boundaries in data are preserved.

Datagram connections closely model the facilities found in many
contemporary packet switched networks such as Ethernet. Broadcast messages may be sent and received.

Establishing Remote Connections

To establish a connection, one side (the server) must execute a **listen()** and and subsequent **accept()** and the other side (the client) a **connect()**. A connection consists of the local socket / remote socket pair. It is therefore possible to have a connection within a single host as long as the local and remote *port* values differ. Each host in an IP network must have at least one host address also known as an IP address. When a host has more than one physical connection to an IP network, it may have more than one IP address. An IP address must be unique within a network. An IP address is 32 bits in length, a port number 16 bits. A value of zero means “any” while a binary value of all
1s means “all.” The latter value is used for broadcasting purposes. Using the **sockaddr** structure conveys the addresses (host/port) to be used in a connection. A local association is performed by the **bind()** function.

Using SOCK_STREAM and SOCK_DGRAM Services

When using the SOCK_STREAM service (TCP), bi-directional data can be sent using the **send()** or **sendto()** functions and received using the **recv()** or **recvfrom()** functions until one side performs a **shutdown(1)** or **shutdown(2)** after which that side cannot send any more data , but can still receive data until the other side performs a **shutdown(1), shutdown(2)** or **closesocket()**.

When using the SOCK_DGRAM service, datagrams can be sent without first establishing a “connection”. In fact UDP provides a “connectionless” service although the connection paradigm is used.

Blocking and Non-blocking Operations

The default behavior of socket functions is to block on an operation and only return when the operation has completed. For example, the **connect()** function only returns after the connection has been performed or an error is encountered. This behavior applies to most socket function calls, such as **recv()** and even **send()**, and especially on SOCK_STREAM connections.

In many, if not most applications, this behavior is unacceptable in the single-threaded DOS environment and must be modified. This modification can be accomplished by making all operations on a socket non-blocking by calling **ioctlsocket()** with the FIONBIO option.
If a non-blocking operation is performed, the function always returns immediately. If the function could not complete without blocking, an error is returned with *errno* containing EWOULDBLOCK. This error should be regarded as a recoverable error and the operation should be retried, preferably at some later time.

Out of band data

TCP "out of band" or urgent data is not implemented. Setting the MSG_OOB flag has no effect in **recv(), recvfrom(), send()** or **sendto()**; it will simply be ignored. The SO_OOBINLINE option will also be ignored and **ioctlsocket()** with the SIOCATMARK command, will always return an argument value of 1.

Error Reporting

In general, the C functions implementing the SOCKETS API return a value of SOCKET_ERROR if the return type is **int** and an error is encountered, in which case, the actual error code is returned in a common variable *errno*. ERR_RE_ENTRY is returned when the SOCKETS kernel has been interrupted. This condition can occur only when the API is called from an interrupt service routine. Programs designed for this type of operation, such as TSR programs activated by a real time clock interrupt, should be coded to handle this error by
re-trying the function at a later stage.

Other sources of Information

Many good books have been written on the Sockets API:

Pocket Guide to TCP/IP Sockets (C Version) by *Michael J. Donahoo, Kenneth L. Calvert*

Windows Sockets Network Programming (Addison-Wesley Advanced Windows Series) by B*ob Quinn, et al; Hardcover*

Internetworking with TCP/IP Vol. III Client-Server Programming

and Applications-Windows Sockets Version by *Douglas E. Comer, David L. Stevens (Contributor) ;Hardcover.*

The Winsock 1.1 help file (WINSOCK.HLP) is also a very useful source of information.

Porting Issues

When porting an application from another BSD Sockets environment like Unix, Linux or Windows (Winsock), a number of issues must be kept in mind. The most important one is that ROM-DOS is a single-user, single-task, single-thread operating system. The use of blocking calls will suspend the system until completion, which may imply an indefinite time under abnormal or even normal conditions. In addition no completion event such as a WSAAsyncSelect windows message for Winsock or a Signal for Unix/Linux is available. Only applications either using nonblocking operations or the **select()** function may be ported successfully. Other applications must be adapted to follow these guidelines.

Unlike Winsock and like BSD Sockets, an error number is returned in the *errno* variable and is only valid directly after an API call. When writing portable code to run on both SAPI and Winsock, a simple
#define can normally be used i.e.

```
#ifdef _Windows
#define Errno WSAGetLastError()
#else
#define Errno errno
#endif
.
.

if (Errno == WSAEWOULDBLOCK)
{
.
.
}
.
.
```

Like in Winsock both the WSAE... of Winsock and the E... error definitions of BSD may be used e.g. WSAEWOULDBLOCK and EWOULDBLOCK. The actual error numbers are the same as that of Winsock, except in cases of DOS error code conflicts e.g. WSAEINVAL has the same value as the DOS EINVAL. Always using the symbolic value and not numeric values, will avoid potential

problems.

The function gethostbyaddr() will always fail with errno == WSANO_DATA.

All the file/socket operations of BSD Sockets must be translated to the *socket() versions as used in Winsock e.g. closesocket() instead of just close().

In Linux/Unix a socket descriptor can be treated the same as a file descriptor; not so for SAPI or Winsock.

For Winsock the WSAStartup() and WSACleanup() functions must be called; make it conditional for portable code.

The "socket set" is defined differently for SAPI/Winsock on the one hand and LINUX/UNIX on the other. Always use the FD_* macros for portable code.

Function Reference

The following sections describe the individual functions of the SOCKETS API.

accept

Syntax:
SOCKET accept (SOCKET *so*, struct sockaddr **psAddress,* int *p
iAddressLen)

Description:
Accepts a connection on a socket.

Parameters

so	A descriptor identifying a socket which is listening for connections after a **listen**().
psAddress	An optional pointer to a buffer which receives the socket address of the connecting peer.
piAddrLen	An optional pointer to an integer which contains the length of the address *psAddress*.

Return Value
If no error occurs, **accept**() returns a value of type SOCKET which is a descriptor for the accepted packet. Otherwise, a value of INVALID_SOCKET is returned, and a specific error code is returned in **errno**.

The integer referred to by *iAddressLen* initially contains the amount of space pointed to by *psAddress*. On return it will contain the actual length in bytes of the socket address returned.

Error Codes

ENETDOWN	The network subsystem has failed.
EFAULT	The *p*iAddressLen* argument is small (less than the sizeof a struct sockaddr).
EINVAL	listen() was not invoked prior to accept().
EMFILE	The queue is empty upon entry to accept() and there are no descriptors available.
ENOBUFS	No buffer space is available.
ENOTSOCK	The descriptor is not a socket.
EOPNOTSUPP	The referenced socket is not a type that supports connection-oriented service.
EWOULDBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.

Remarks

This function extracts the first connection on the queue of pending connections on listening socket *so*, creates a new socket with the same properties as *so* and returns a handle to the new socket. If no
pending connections are present on the queue, and the socket is not marked as non-blocking, **accept()** blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, **accept()** returns an error as described below. Socket *so* remains listening.

The argument *psAddress* is a result parameter that is filled in with the socket address of the connecting peer. The p*iAddressLen* is a value-result parameter; it should initially contain the amount of space pointed to by *psAddress*; on return it will contain the actual length (in bytes) of the socket address returned. This call is used with the connectionbased SOCK_STREAM socket type. If *psAddress* and/or p *iAddressLen* are equal to NULL, then no information about the remote peer socket address of the accepted socket is returned.

See Also

bind(), connect(), listen(), select(), socket()

bind

Syntax:
int bind (SOCKET *so,* const struct sockaddr * *psAddress*, int *iAddressLen*)

Description:
Associates a local socket address with a socket.

Parameters

so A descriptor identifying an unbound socket.

psAddress The socket address to assign to the socket. The sockaddr structure is defined as follows:

```
struct sockaddr
{
        u_short sa_family;
        char sa_data[14];
};
```

iAddressLen The length of the name *psAddress*.

Return Value
If no error occurs, **bind()** returns 0. Otherwise, it returns SOCKET_ERROR, and a specific error code is returned in **errno**.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EADDRINUSE	The specified address is already in use. (See the SO_REUSEADDR socket option under setsockopt().)
EFAULT	The *iAddressLen* argument is too small (less than the size of a struct sockaddr).
EAFNOSUPPORT	The specified address family is not supported by this protocol.
EINVAL	The socket is already bound to an address.
ENOBUFS	Not enough buffers available, too many connections.
ENOTSOCK	The descriptor is not a socket.

Remarks

This routine is used on an unconnected datagram or stream socket, before subsequent **connect**()s or **listen**()s. When a socket is created with **socket**(), it exists in a name space (address family), but it has no socket address assigned. **bind**() establishes the local association
(host address/port number) of the socket by assigning a local address to an unnamed socket.

In the Internet address family, an address consists of several components. For SOCK_DGRAM and SOCK_STREAM, the address consists of three parts: a host address, the protocol number (set implicitly to UDP or TCP, respectively), and a port number which identifies the application. If an application does not care what address
is assigned to it, it may specify an Internet address equal to INADDR_ANY, a port equal to 0, or both. If the Internet address is equal to INADDR_ANY, any appropriate network interface will be used; this simplifies application programming in the presence of multihomed hosts. If the port is specified as 0, SOCKETS will assign a unique port to the application. The application may use **getsockname()** after **bind()** to learn the address that has been assigned to it, but note that **getsockname()** will not necessarily fill in the Internet address until the socket is connected, since several Internet addresses may be valid if the host is multi-homed.

See Also

connect(), listen(), getsockname(), setsockopt(), socket().

closesocket

Syntax:
int closesocket (SOCKET *so*)

Description:
Closes a socket.

Parameters	**Description**
so	A descriptor identifying a socket.

Return Value
If no error occurs, **closesocket()** returns 0. Otherwise, a value of
SOCKET_ERROR is returned, and a specific error code is returned in errno**.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
ENOTSOCK	The descriptor is not a socket.
EWOULDBLOCK	The socket is marked as nonblocking and SO_LINGER is set to a nonzero timeout value.

Remarks
This function closes a socket. More precisely, it releases the socket descriptor *so*, so that further references to *so* will fail with the error ENOTSOCK. If this is the last reference to the underlying socket, the
associated naming information and queued data are discarded.

The semantics of **closesocket**() are affected by the socket options
SO_LINGER and SO_DONTLINGER as follows:

Option	Interval	Type of close	Wait for close?
SO_DONTLINGER	Don't care	Graceful	No
SO_LINGER	Zero	Hard	No
SO_LINGER	Non-zero	Graceful	Yes

If SO_LINGER is set (i.e. the *l_onoff* field of the linger structure is non-zero) with a zero timeout interval (*l_linger* is zero), **closesocket()** is not blocked even if queued data has not yet been sent or acknowledged. This is called a "hard" or "abortive" close, because the
socket's virtual circuit is reset immediately, and any unsent data is lost. Any **recv()** call on the remote side of the circuit will fail with
ECONNRESET.

If SO_LINGER is set with a non-zero timeout interval, the **closesocket()** call blocks until the remaining data has been sent or until the timeout expires. This is called a graceful disconnect. Note that if the socket is set to non-blocking and SO_LINGER is set to a non-zero timeout, the call to **closesocket()** will fail with an error of EWOULDBLOCK.

If SO_DONTLINGER is set on a stream socket (i.e. the *l_onoff* field of the linger structure is zero), the **closesocket()** call will return immediately. However, any data queued for transmission will be sent
if possible before the underlying socket is closed. This is also called a graceful disconnect. Note that in this case SOCKETS may not release the socket and other resources for an arbitrary period, which may
affect applications which expect to use all available sockets.

See Also
accept(), socket(), ioctlsocket(), setsockopt.

connect

Syntax:
int connect (SOCKET *so*, const struct sockaddr * *psAddress*, int *iAddressLen*)

Description:
Establishes a connection to a peer.

Parameters	Description
so	A descriptor identifying an unconnected socket.
psAddress	The socket address of the peer to which the socket is to be connected.
iAddressLen	The length of *psAddress*.

Return Value
If no error occurs, **connect()** returns 0. Otherwise, it returns SOCKET_ERROR, and a specific error code is returned in errno**.**

On a blocking socket, the return value indicates success or failure of the connection attempt.

On a non-blocking socket, if the return value is SOCKET_ERROR and **errno** indicates an error code of EWOULDBLOCK, then your application can either:

1. Use **select()** to determine the completion of the connection request by checking if the socket is writeable, or
2. Use **recv()** until either no error or an error of EWOULDBLOCK is returned.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EADDRINUSE	The specified address is already in use.
EADDRNOTAVAIL	The specified address is not available from the local machine.
EAFNOSUPPORT	Addresses in the specified family cannot be used with this socket.
ECONNREFUSED	The attempt to connect was forcefully

rejected.

EDESTADDREQ	A destination address is required.
EFAULT	The *iAddressLen* argument is incorrect.
EINVAL	The socket is not already bound to an address.
EISCONN	The socket is already connected.
EMFILE	No more file descriptors are available.
ENETUNREACH	The network can't be reached from this host at this time.
ENOBUFS	No buffer space is available. The socket cannot be connected.
ENOTSOCK	The descriptor is not a socket.
ETIMEDOUT	Attempt to connect timed out without establishing a connection
EWOULDBLOCK	The socket is marked as non-blocking and the connection cannot be completed immediately. It is possible to select() the socket while it is connecting by select()ing it for writing.

Remarks

This function is used to create a connection to the specified foreign socket address. The parameter *so* specifies an unconnected datagram or stream socket. If the socket is unbound, unique values are assigned to the local association by the system, and the socket is marked as bound. Note that if the address field of the *psAddress* structure is all zeroes, **connect()** will return the error EADDRNOTAVAIL.

For stream sockets (type SOCK_STREAM), an active connection is initiated to the foreign host using *psAddress* (an address in the name space of the socket). When the socket call completes successfully, the socket is ready to send/receive data.

For a datagram socket (type SOCK_DGRAM), a default destination is set, which will be used on subsequent **send**() and **recv**() calls.

See Also

accept(), bind(), getsockname(), socket() and select().

getpeername

Syntax:
int getpeername (SOCKET *so*, struct sockaddr * *psAddress*, int *
piAddressLen)

Description:
Gets the socket address of the peer to which a socket is connected.

Parameters	Description
so	A descriptor identifying a connected socket.
psAddress	The structure which is to receive the socket address of the peer.
piAddressLen	A pointer to the size of the *psAddress* structure.

Return Value
If no error occurs, **getpeername()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code is returned in **errno.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EFAULT	The **piAddressLen* argument is not large enough.
ENOTCONN	The socket is not connected.
ENOTSOCK	The descriptor is not a socket.

Remarks
getpeername() retrieves the socket address of the peer connected to the socket *so* and stores it in the struct sockaddr identified by *psAddress*. It is used on a connected datagram or stream socket.

On return, the *piAddressLen* argument contains the actual size of the socket address returned in bytes.

See Also
bind(), socket(), getsockname().

getsockname

Syntax:
int getsockname (SOCKET *so*, struct sockaddr * *psAddress*, int *
p*iAddressLen*)

Description:
Gets the local socket address for a socket.

Parameters	**Description**
so	A descriptor identifying a bound socket.
psAddress	Receives the socket address (name) of the socket.
*p*iAddressLen	A pointer to the size of the *psAddress* buffer.

Return Value
If no error occurs, **getsockname()** returns 0. Otherwise, a value of
SOCKET_ERROR is returned, and a specific error code is returned in **errno.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EFAULT	The **piAddressLen* argument is not large enough.
ENOTSOCK	The descriptor is not a socket.
EINVAL	The socket has not been bound to an address with bind().

Remarks
getsockname() retrieves the current socket address for the specified socket descriptor in *psAddress*. It is used on a bound and/or
connected socket specified by the *so* parameter.The local association is returned. This call is especially useful when a **connect()** call has been made without doing a **bind()** first; this call provides the only means by which you can determine the local association which has been set by the system.

On return, the *piAddressLen* argument contains the actual size of the socket address returned in bytes.If a socket was bound to INADDR_ANY, indicating that any of the host's IP addresses should be used for the socket, **getsockname()** will not necessarily return
information about the host IP address, unless the socket has been
connected with **connect()** or **accept().** A SOCKETS application must not assume that the IP address will be changed from INADDR_ANY

unless the socket is connected. This is because for a multi-homed host the IP address that will be used for the socket is unknown unless the socket is connected.

See Also
bind(), socket(), getpeername().

getsockopt

Syntax:
int getsockopt (SOCKET *so*, int *iLevel*, int *iOptname*, char * pc*Optval*, int * *piOptlen*)

Description:
Retrieves a socket option.

Parameters	**Description**
so	A descriptor identifying a socket.
iLevel	The level at which the option is defined; the only supported levels are SOL_SOCKET and IPPROTO_TCP.
iOptname	The socket option for which the value is to be retrieved.
pcOptval	A pointer to the buffer in which the value for the requested option is to be returned.
piOptlen	A pointer to the size of the *pcOptval* buffer.

Return Value
If no error occurs, **getsockopt()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EFAULT	The *piOptlen* argument was invalid.
ENOPROTOOPT	The option is unknown or unsupported. In particular, SO_BROADCAST is not supported on sockets of type SOCK_STREAM, while SO_ACCEPTCONN, SO_DONTLINGER, SO_KEEPALIVE, SO_LINGER and SO_OOBINLINE are not supported on sockets of type SOCK_DGRAM.
ENOTSOCK	The descriptor is not a socket.

Remarks

getsockopt() retrieves the current value for a socket option associated with a socket of any type, in any state, and stores the result in pcO*ptval*. Options may exist at multiple protocol levels, but they are always present at the uppermost "socket" level. Options
affect socket operations, such as whether an operation blocks or not, the routing of packets, out -of-band data transfer, etc.

The value associated with the selected option is returned in the buffer pcO*ptval*. The integer pointed to by piO*ptlen* should originally contain the size of this buffer; on return, it will be set to the size of the value returned. For SO_LINGER, this will be the size of a struct linger; for all other options it will be the size of an integer.

If the option was never set with **setsockopt(),** then **getsockopt()** returns the default value for the option.The following options are supported for **getsockopt()**. The Type identifies the type of data addressed by *optval*. The TCP_NODELAY option uses *level* IPPROTO_TCP; all other options use *level* SOL_SOCKET.

Value	Type	Meaning	Default
SO_ACCEPTCONN	BOOL	Socket is listen()ing.	FALSE
SO_BROADCAST	BOOL	Socket is configured for the transmission of broadcast messages.	FALSE
SO_DEBUG	BOOL	Debugging is enabled.	FALSE
SO_DONTLINGER	BOOL	If true, the SO_LINGER option is disabled.	TRUE
SO_DONTROUTE	BOOL	Routing is disabled.	FALSE
SO_ERROR	int	Retrieve error status and clear.	0
SO_KEEPALIVE	BOOL	Keepalives are being sent.	FALSE
SO_LINGER	struct linger*	Returns the current linger options.	l_onoff is 0
SO_OOBINLINE	BOOL	Out-of-band data is being received in the normal data stream.	FALSE
SO_RCVBUF	int	Buffer size for receives	1460
SO_REUSEADDR	BOOL	The socket may be bound to an address which is already in use.	FALSE
SO_SNDBUF	int	Buffer size for sends	1460
SO_TYPE	int	The type of the socket (e.g. SOCK_STREAM).	As created
TCP_NODELAY	BOOL	Disables the Nagle algorithm for send coalescing.	FALSE

Calling **getsockopt()** with an unsupported option will result in an error code of ENOPROTOOPT being returned from **WSAGetLastError()**.

See Also
setsockopt(), socket().

htonl

Syntax:
u_long htonl (u_long *ulHostlong*)

Description:
Converts a **u_long** from host to network byte order.

Parameters	**Description**
ulHostlong	A 32-bit number in host byte order.

Return Value
htonl() returns the value in network byte order.

Remarks
This routine takes a 32-bit number in host byte order and returns a 32-bit number in network byte order.

See Also
htons(), ntohl(), ntohs().

htons

Syntax
u_short htons (u_short usH*ostshort*)

Description:
Converts a **u_short** from host to network byte order.

Parameters	**Description**
sHostshort	A 16-bit number in host byte order.

Return Value
htons() returns the value in network byte order.

Remarks
This routine takes a 16-bit number in host byte order and returns a 16-bit number in network byte order.

See Also
htonl(), ntohl(), ntohs().

inet_addr

Syntax:
unsigned long inet_addr (const char * *pc*)

Description:
Converts a string containing a dotted address into an **in_addr**.

Parameters	**Description**
pc	A character string representing a number expressed in the Internet standard "." notation.

Return Value
If no error occurs, **inet_addr()** returns an unsigned long containing a suitable binary representation of the Internet address given. If the passed-in string does not contain a legitimate Internet address, for example if a portion of an "a.b.c.d" address exceeds 255, **inet_addr()** returns the value INADDR_NONE.

Remarks
This function interprets the character string specified by the *pc* parameter. This string represents a numeric Internet address expressed in the Internet standard "." notation. The value returned is a number suitable for use as an Internet address. All Internet addresses are returned in network order (bytes ordered from left to right).

Internet Addresses

Values specified using the "." notation take one of the following forms:

a.b.c.d a.b.c a.b a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the Intel architecture, the bytes referred to above appear as "d.c.b.a". That is, the bytes on an Intel processor are ordered from right to left.

Note: The following notations are only used by Berkeley, and nowhere

else on the Internet. In the interests of compatibility with their software, they are supported as specified.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as "128.net.host".

When a two part address is specified, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as "net.host".

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

See Also
inet_ntoa()

inet_ntoa

Syntax:
char * inet_ntoa (struct in_addr *sIn*)

Description:
Converts a network address into a string in dotted format.

Parameters	**Description**
sIn	A structure which represents an Internet host address.

Return Value
If no error occurs, **inet_ntoa()** returns a char pointer to a static buffer containing the text address in standard "." notation. Otherwise, it returns NULL. The data should be copied before another SOCKETS call is made.

Remarks
This function takes an Internet address structure specified by the *sIn* parameter. It returns an ASCII string representing the address in "." notation as "a.b.c.d". Note that the string returned by **inet_ntoa()** resides in memory which is allocated by SOCKETS. The application should not make any assumptions about the way in which the memory is allocated. The data is guaranteed to be valid until the next SOCKETS API call, but no longer.

See Also
inet_addr().

ioctlsocket

Syntax:
int ioctlsocket (SOCKET *so*, long *lCmd*, u_long * *pulArgp*)

Description:
Controls the mode of a socket.

Parameters	**Description**
so	A descriptor identifying a socket.
lCmd	The command to perform on the socket *so*.
pulArgp	A pointer to a parameter for *lCmd*.

Return Value
Upon successful completion, the **ioctlsocket()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code is
returned in errno**.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EINVAL	*lCmd* is not a valid command, or pulA*rgp* is not an acceptable parameter for *lCmd*, or the command is not applicable to the type of socket supplied
ENOTSOCK	The descriptor *so* is not a socket.

Remarks
This routine may be used on any socket in any state. It is used to get or retrieve operating parameters associated with the socket, independent of the protocol and communications subsystem. The following commands are supported:

Command	**Semantics**
FIONBIO	Enable or disable non-blocking mode on the socket *so*. *pulArgp* points at an **unsigned long**, which is non-zero if non-blocking mode is to be enabled and zero if it is to be disabled. When a socket is created, it operates in blocking mode (i.e. non-blocking mode is disabled). This is consistent

with BSD sockets.

FIONREAD Determine the amount of data which can be read atomically from socket *so*. *pulArgp* points at an **unsigned long** in which **ioctlsocket()** stores the result. If *so* is of type SOCK_STREAM, FIONREAD returns the total amount of data which may be read in a single **recv()**; this is normally the same as the total amount of data queued on the socket. If *so* is of type SOCK_DGRAM, FIONREAD returns the size of the first datagram queued on the socket.

SIOCATMARK Determine whether or not all out-of-band data has been read. This applies only to a socket of type SOCK_STREAM which has been configured for in-line reception of any out-of-band data (SO_OOBINLINE). If no out-of-band data is waiting to be read, the operation returns TRUE. Otherwise it returns FALSE, and the next **recv()** or **recvfrom()** performed on the socket will retrieve some or all of the data preceding the "mark"; the application should use the SIOCATMARK operation to determine whether any remains. If there is any normal data preceding the "urgent" (out of band) data, it will be received in order. (Note that a **recv()** or **recvfrom()** will never mix out-of-band and normal data in the same call.) *argp* points at a **BOOL** in which **ioctlsocket()** stores the result.

Compatibility

This function is a subset of **ioctl()** as used in Berkeley sockets. In particular, there is no command which is equivalent to FIOASYNC, while SIOCATMARK is the only socketlevel command which is supported.

See Also

socket(), setsockopt(), getsockopt().

listen

Syntax:
int listen (SOCKET *so*, int *iBacklog*)

Description:
Establishes a socket to listen for incoming connection.

Parameters	**Description**
so	A descriptor identifying a bound, unconnected socket.
iBacklog	The maximum length to which the queue of pending connections may grow.

Return Value
If no error occurs, **listen()** returns 0. Otherwise, a value of
SOCKET_ERROR is returned, and a specific error code is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EADDRINUSE	An attempt has been made to listen() on an address in use.
EINVAL	The socket has not been bound with bind() or is already connected.
EISCONN	The socket is already connected.
EMFILE	No more file descriptors are available.
ENOBUFS	No buffer space is available.
ENOTSOCK	The descriptor is not a socket.
EOPNOTSUPP	The referenced socket is not of a type that supports the listen() operation.

Remarks
To accept connections, a socket is first created with **socket**(), a backlog for incoming connections is specified with **listen**(), and then the connections are accepted with **accept**(). **listen**() applies only to
sockets that support connections, i.e. those of type SOCK_STREAM. The socket *so* is put into "passive" mode where incoming connections are acknowledged and queued pending acceptance by the process.
This function is typically used by servers that could have more than one connection request at a time: if a connection request arrives with the queue full, the client will receive an error with an

indication of
ECONNREFUSED.

Compatibility

iBacklog is limited (silently) to 5. As in 4.3BSD, illegal values (less than 1 or greater than 5) are replaced by the nearest legal value.

See Also

accept(), connect(), socket().

ntohl

Syntax:
u_long ntohl (u_long *ulNetlong*)

Description:
Converts a **u_long** from network to host byte order.

Parameters	**Description**
ulNetlong	A 32-bit number in network byte order.

Return Value
ntohl() returns the value in host byte order.

Remarks
This routine takes a 32-bit number in network byte order and returns a 32-bit number in host byte order.

See Also
htonl(), htons(), ntohs().

ntohs

Syntax
u_short ntohs (u_short *usNetshort*)

Description:
Converts a **u_short** from network to host byte order.

Return Value
ntohs() returns the value in host byte order.

Parameters	**Description**
usNetshort	A 16-bit number in network byte order.

Remarks
This routine takes a 16-bit number in network byte order and returns a 16-bit number in host byte order.

See Also
htonl(), htons(), ntohl().

recv

Syntax:
int recv (SOCKET *so*, char * pc*buf*, int *iLen*, int *iFlags*)

Description:
Receives data from a socket.

Parameters	Description
so	A descriptor identifying a connected socket.
pcBuf	A buffer for the incoming data.
iLen	The length of pcB*uf*.
iFlags	Specifies the way in which the call is made.

Return Value
If no error occurs, **recv()** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, a value of
SOCKET_ERROR is returned, and a specific error code is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
ENOTCONN	The socket is not connected.
ENOTSOCK	The descriptor is not a socket.
EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
ESHUTDOWN	The socket has been shutdown; it is not possible to recv() on a socket after shutdown() has been invoked with *how* set to 0 or 2.
EWOULDBLOCK	The socket is marked as non-blocking and the receive operation would block.
EMSGSIZE	The datagram was too large to fit into the specified buffer and was truncated.
EINVAL	The socket has not been bound with bind().
ECONNABORTED	The virtual circuit was aborted due to timeout or other failure.
ECONNRESET	The virtual circuit was reset by the remote side.

Remarks

This function is used on connected datagram or stream sockets specified by the *so* parameter and is used to read incoming data.

For sockets of type SOCK_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (socket option SO_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the **ioctlsocket()** SIOCATMARK to determine whether any more out-of- band data remains to be read.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the datagram, the excess data is lost, and **recv()** returns the error EMSGSIZE.

If no incoming data is available at the socket, the **recv()** call waits for data to arrive unless the socket is non-blocking. In this case a value of SOCKET_ERROR is returned with the error code set to EWOULDBLOCK. The **select**() call may be used to determine when more data arrives.

If the socket is of type SOCK_STREAM and the remote side has shut down the connection gracefully, a **recv()** will complete immediately with 0 bytes received. If the connection has been reset, a **recv()** will fail with the error ECONNRESET.

iFlags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the i*Flags* parameter. The latter is constructed by or-ing any of the following values:

Value	**Meaning**
MSG_	PEEK Peek at the incoming data. The data is copied into the buffer but is not removed from the input queue.
MSG_OOB	Process out-of-band data.

See Also

recvfrom(), ,recv(), send(), select(), socket()

recvfrom

Syntax
int recvfrom (SOCKET *so*, char * pcB*uf*, int iL*en*, int iF*lags*, struct sockaddr * psF*rom*, int * piF*romlen*)

Description:
Receives a datagram and store the source address.

Parameters	Description
so	A descriptor identifying a bound socket.
pcBuf	A buffer for the incoming data.
iLen	The length of pcB*uf*.
iFlags	Specifies the way in which the call is made.
psFrom	An optional pointer to a buffer which will hold the source address upon return.
piFromlen	An optional pointer to the size of the psF*rom* buffer.

Return Value
If no error occurs, **recvfrom()** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code is returned in
errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EFAULT	The piF*romlen* argument was invalid: the psF*rom* buffer was too small to accommodate the peer address.
EINVAL	The socket has not been bound with bind().
ENOTCONN	The socket is not connected (SOCK_STREAM only).
ENOTSOCK	The descriptor is not a socket.
EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
ESHUTDOWN	The socket has been shutdown; it is not possible to recvfrom() on a socket after shutdown() has been invoked with *how* set to 0 or 2.
EWOULDBLOCK	The socket is marked as non-blocking

and the recvfrom() operation would block.

EMSGSIZE	The datagram was too large to fit into the specified buffer and was truncated.
ECONNABORTED	The virtual circuit was aborted due to timeout or other failure.
ECONNRESET	The virtual circuit was reset by the remote side.

Remarks

This function is used to read incoming data on a (possibly connected) socket and capture the address from which the data was sent.

For sockets of type SOCK_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (socket option SO_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the **ioctlsocket()** SIOCATMARK to determine whether any more out-of- band data remains to be read. The psF*rom* and piF*romlen* parameters are ignored for SOCK_STREAM sockets.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the message, the excess data is lost, and **recvfrom()** returns the error code EMSGSIZE.

If psF*rom* is non-zero, and the socket is of type SOCK_DGRAM, the network address of the peer which sent t he data is copied to the corresponding struct sockaddr. The value pointed to by piF*romlen* is initialized to the size of this structure, and is modified on return to indicate the actual size of the address stored there.

If no incoming data is available at the socket, the **recvfrom()** call waits for data to arrive unless the socket is non-blocking. In this case a value of SOCKET_ERROR is returned with the error code set to EWOULDBLOCK. The **select**() call may be used to determine when more data arrives.

If the socket is of type SOCK_STREAM and the remote side has shut down the connection gracefully, a **recvfrom()** will complete immediately with 0 bytes received. If the connection has been reset **recv()** will fail with the error ECONNRESET.

iFlags may be used to influence the behavior of the function
invocation beyond the options specified for the associated socket. That is, the semantics of this function aredetermined by the socket options and the i*Flags* parameter. The latter is constructed by or-ing
any of the following values:

Value	Meaning
MSG_PEEK	Peek at the incoming data. The data is copied into the buffer but is not removed from the input queue.
MSG_OOB	Process out-of-band data.

See Also
recv(), send(), socket().

select

Syntax:
int select (int iN*fds*, fd_set * psR*eadfds*, fd_set * psW*ritefds*, fd_set * psE*xceptfds*, const struct timeval * psT*imeout*)

Description:
Determines the status of one or more sockets, waiting if necessary.

Parameters	Description
iNfds	This argument is ignored and included only for the sake of compatibility.
psReadfds	An optional pointer to a set of sockets to be checked for readability.
psWritefds	An optional pointer to a set of sockets to be checked for writability
psExceptfds	An optional pointer to a set of sockets to be checked for errors.
psTimeout	The maximum time for **select()** to wait, or NULL for blocking operation.

Return Value
select() returns the total number of descriptors which are ready and contained in the fd_set structures, 0 if the time limit expired, or SOCKET_ERROR if an error occurred. If the return value is SOCKET_ERROR, **errno** contains the specific error code.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EINVAL	The psT*imeout* value is not valid.
ENOTSOCK	One of the descriptor sets contains an entry which is not a socket.

Remarks

This function is used to determine the status of one or more sockets. For each socket, the caller may request information on read, write or error status. The set of sockets for which a given status is requested is indicated by an fd_set structure. Upon return, the structure is updated to reflect the subset of these sockets which meet the specified condition, and **select()** returns the number of sockets meeting the conditions. A set of macros is provided for manipulating an fd_set. These macros are compatible with those used in the Berkeley software, but the underlying representation is completely different and the same as that used in Winsock.

The parameter psR*eadfds* identifies those sockets which are to be checked for readability. If the socket is currently **listen()**ing, it will be marked as readable if an incoming connection request has been received, so that an **accept()** is guaranteed to complete without blocking. For other sockets, readability means that queued data is available for reading or, for sockets of type SOCK_STREAM, that the virtual socket corresponding to the socket has been closed, so that a **recv()** or **recvfrom()** is guaranteed to complete without blocking. If the virtual circuit was closed gracefully, then a **recv()** will return immediately with 0 bytes read; if the virtual circuit was reset, then a **recv()** will complete immediately with the error code ECONNRESET. The presence of out-of-band data will be checked if the socket option SO_OOBINLINE has been enabled (see **setsockopt()**).

The parameter psW*ritefds* identifies those sockets which are to be checked for writability. If a socket is **connect()**ing (non-blocking), writability means that the connection establishment successfully completed. If the socket is not in the process of **connect()**ing, writability means that a **send()** or **sendto()** will complete without blocking.

The parameter psE*xceptfds* identifies those sockets which are to be checked for the presence of out-of-band data or any exceptional error conditions. Note that out -of-band data will only be reported in this way if the option SO_OOBINLINE is FALSE. For a SOCK_STREAM, the breaking of the connection by the peer or due to KEEPALIVE failure will be indicated as an exception. If a socket is **connect()**ing (non- blocking), failure of the connect attempt is indicated in *psExceptfds*.

Any of psR*eadfds*, psW*ritefds*, or psE*xceptfds* may be given as NULL if no descriptors are of interest.

Four macros are defined in the header file **socket.h** for manipulating the descriptor sets. The variable FD_SETSIZE determines the maximum number of descriptors in a set. (The default value of FD_SETSIZE is 16, which may be modified by #defining FD_SETSIZE to another value before #including **socket.h**.) Internally, an fd_set is represented as an array of SOCKETs. The macros are:

FD_CLR(so, *psSet)	Removes the descriptor so from set.
FD_ISSET(so, *pSset)	Nonzero if so is a member of the set, zero otherwise.
FD_SET(so, *psSet)	Adds descriptor so to set.
FD_ZERO(*psSet)	Initializes the set to the NULL set.

The parameter psT*imeout* controls how long the **select()** may take to comp lete. If psT*imeout* is a null pointer, **select()** will block indefinitely until at least one descriptor meets the specified criteria. Otherwise, psT*timeout* points to a struct timeval which specifies the maximum time that **select()** should wait before returning. If the timeval is initialized to {0, 0}, **select()** will return immediately; this is used to "poll" the state of the selected sockets.

See Also
accept(), connect(), recv(), recvfrom(), send().

send

Syntax:
int send (SOCKET *so*, const char * *pcBuf*, int *iLen*, int *iFlags*)

Description:
Sends data on a connected socket.

Parameters	**Description**
so	A descriptor identifying a connected socket.
pcBuf	A buffer containing the data to be transmitted.
iLen	The length of the data in *pcBuf*.
iFlags	Specifies the way in which the call is made.

Return Value
If no error occurs, **send()** returns the total number of characters sent. (Note that this may be less than the number indicated by *len*.)
Otherwise, a value of SOCKET_ERROR is returned, and a specific error code is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EACCES	The requested address is a broadcast address, but the appropriate flag was not set.
EFAULT	The *pcBuf* argument is not in a valid part of the user address space.
ENETRESET	The connection must be reset because SOCKETS dropped it.
ENOBUFS	SOCKETS reports a buffer deadlock.
ENOTCONN	The socket is not connected.
ENOTSOCK	The descriptor is not a socket.
EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.
ESHUTDOWN	The socket has been shutdown; it is not possible to send() on a socket after shutdown() has been invoked with how set to 1 or 2.
EWOULDBLOCK	The socket is marked as non-blocking

	and the requested operation would block.
EMSGSIZE	The socket is of type SOCK_DGRAM, and the datagram is larger than the maximum supported by SOCKETS.

EINVAL	The socket has not been bound with bind().
ECONNABORTED	The virtual circuit was aborted due to timeout or other failure.
ECONNRESET	The virtual circuit was reset by the remote side.

Remarks

send() is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets. If the data is too long to pass atomically through the underlying protocol the error EMSGSIZE is returned, and no data is transmitted.

Note that the successful completion of a **send()** does not indicate that the data was successfully delivered.

If no buffer space is available within the transport system to hold the data to be transmitted, **send**() will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking SOCK_STREAM sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The **select**() call may be used to determine when it
is possible to send more data.

iFlags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *flags* parameter. The latter is constructed by oring any
of the following values:

Value	**Meaning**
MSG_DONTROUTE	Specifies that the data should not be subject to routing
MSG_OOB	Send out-of-band data (SOCK_STREAM only)

See Also

recv(), recvfrom(), socket(), sendto().

sendto

Syntax:
int sendto (SOCKET *so*, const char * *pcBuf*, int *iLen*, int *iFlags*, const struct sockaddr * *psTo*, int *iTolen*)

Description:
Sends data to a specific destination.

Parameters	Description
so	A descriptor identifying a socket.
pcBuf	A buffer containing the data to be transmitted.
iLen	The length of the data in *pcBuf*.
iFlags	Specifies the way in which the call is made.
PsTo	An optional pointer to the address of the target socket.
iITolen	The size of the address in *to*.

Return Value
If no error occurs, **sendto()** returns the total number of characters sent. (Note that this may be less than the number indicated by *len*.) Otherwise, a value of SOCKET_ERROR is returned, and a specific
error code is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EACCES	The requested address is a broadcast address, but the appropriate flag was not set.
EFAULT	The pcBuf or psTo parameters are not part of the user address space, or the psTo argument is too small (less than the sizeof a struct sockaddr).
ENETRESET	The connection must be reset because SOCKETS dropped it.
ENOBUFS	SOCKETS reports a buffer deadlock.
ENOTCONN	The socket is not connected (SOCK_STREAM only).
ENOTSOCK	The descriptor is not a socket.
EOPNOTSUPP	MSG_OOB was specified, but the socket is not of type SOCK_STREAM.

ESHUTDOWN	The socket has been shutdown; it is not possible to sendto() on a socket after shutdown() has been invoked with how set to 1 or 2.
EWOULDBLOCK	The socket is marked as non-blocking and the requested operation would block.
EMSGSIZE	The socket is of type SOCK_DGRAM, and the datagram is larger than the maximum supported by SOCKETS.
ECONNABORTED	The virtual circuit was aborted due to timeout or other failure.
ECONNRESET	The virtual circuit was reset by the remote side.
EADDRNOTAVAIL	The specified address is not available from the local machine.
EAFNOSUPPORT be	Addresses in the specified family cannot used with this socket.
EDESTADDRREQ	A destination address is required.
ENETUNREACH	The network can't be reached from this host at this time.

Remarks

sendto() is used on datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets. If the data is too long to pass atomically through the underlying protocol the error EMSGSIZE is returned, and no data is transmitted.

Note that the successful completion of a **sendto()** does not indicate that the data was successfully delivered.

sendto() is normally used on a SOCK_DGRAM socket to send a datagram to a specific peer socket identified by the psT*o* parameter. On a SOCK_STREAM socket, the psT*o* and iT*olen* parameters are ignored; in this case the **sendto()** is equivalent to **send**().

To send a broadcast (on a SOCK_DGRAM only), the address in the *to* parameter should be constructed using the special IP address INADDR_BROADCAST (defined in **socket.h**) together with the intended port number. It is generally inadvisable for a broadcast datagram to exceed the size at

which fragmentation may occur, which implies that the data portion of the datagram (excluding headers) should not exceed 512 bytes.

If no buffer space is available within the transport system to hold the data to be transmitted, **sendto**() will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking SOCK_STREAM sockets, the number of bytes written may be between 1 and the
requested length, depending on buffer availability on both t he local
and foreign hosts. The **select**() call may be used to determine when it is possible to send more data.

iFlags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *iFlags* parameter. The latter is constructed by or-ing any of the following values:

Value	**Meaning**
MSG_DONTROUTE	Specifies that the data should not be subject to routing.
MSG_OOB	Send out-of-band data (SOCK_STREAM only Out of band data)

See Also
recv(), recvfrom(), socket(), send().

setsockopt

Syntax:
int setsockopt (SOCKET *so*, int *level*, int *optname*, const char * *optval*, int *optlen*)

Description:
Sets a socket option.

Parameters	**Description**
so	A descriptor identifying a socket.
level	The level at which the option is defined; the only supported levels are SOL_SOCKET and IPPROTO_TCP.
optname	The socket option for which the value is to be set.
optval	A pointer to the buffer in which the value for the requested option is supplied.
optlen	The size of the optval buffer.

Return Value
If no error occurs, **setsockopt()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EFAULT	*optval* is not in a valid part of the process address space.
EINVAL	*level* is not valid, or the information in *optval* is not valid.
ENETRESET	Connection has timed out when SO_KEEPALIVE is set.
ENOPROTOOPT	The option is unknown or unsupported. In particular, SO_BROADCAST is not supported on sockets of type SOCK_STREAM, while SO_DONTLINGER, SO_KEEPALIVE, SO_LINGER and SO_OOBINLINE are not supported on sockets of type SOCK_DGRAM.
ENOTCONN	Connection has been reset when SO_KEEPALIVE is set.

ENOTSOCK	The descriptor is not a socket.

Remarks

setsockopt() sets the current value for a socket option associated with a socket of any type, in any state. Although options may exist at multiple protocol levels, this specification only defines options that
exist at the uppermost "socket" level. Options affect socket operations, such as whet her expedited data is received in the normal data stream, whether broadcast messages may be sent on the socket, etc.

There are two types of socket options: Boolean options that enable or disable a feature or behavior, and options which require an integer value or structure. To enable a Boolean option, *optval* points to a nonzero integer. To disable the option *optval* points to an integer equal to zero. *optlen* should be equal to sizeof(int) for Boolean options. For other options, *optval* points to the an integer or structure that contains the desired value for the option, and *optlen* is the length of the integer or structure.

SO_LINGER controls the action taken when unsent data is queued on a socket and a **closesocket()** is performed. See **closesocket()** for a description of the way in which the SO_LINGER settings affect the semantics of **closesocket()**. The application sets the desired behavior
by creating a *struct linger* (pointed to by the *optval* argument) with the following elements:

```
struct linger
{
        int l_onoff;
        int l_linger;
}
```

To enable SO_LINGER, the application should set *l_onoff* to a non- zero value, set *l_linger* to 0 or the desired timeout (in seconds), and call **setsockopt()**. To enable SO_DONTLINGER (i.e. disable SO_LINGER) *l_onoff* should be set to zero and **setsockopt()** should be called.

By default, a socket may not be bound (see **bind()**) to a local address which is already in use. On occasions, however, it may be desirable to "re-use" an address in this way. Since every connection is uniquely identified by the combination of local and remote addresses, there is no problem with having two sockets bound to the same local address as long as the remote addresses are different. To inform SOCKETS that a **bind()** on a socket should not be disallowed because the desired address is already in use by another socket, the application should set the SO_REUSEADDR

socket option for the socket before

issuing the **bind()**. Note that the option is interpreted only at the time of the **bind()**: it is therefore unnecessary (but harmless) to set the option on a socket which is not to be bound to an existing address, and setting or resetting the option after the **bind()** has no effect on this
or any other socket.

An application may request that SOCKETS enable the use of "keep-alive" packets on TCP connections by turning on the SO_KEEPALIVE socket option. If a connection is dropped as the result of "keep -alives" the error code ENETRESET is returned to any calls in progress on the socket, and any subsequent calls will fail with ENOTCONN.

The TCP_NODELAY option disables the Nagle algorithm. The Nagle algorithm is used to reduce the number of small packets sent by a host by buffering unacknowledged send data until a full-size packet can be sent. However, for some applications this algorithm can impede performance, and TCP_NODELAY may be used to turn it off. Application writers should not set TCP_NODELAY unless the impact
of doing so is well-understood and desired, since setting TCP_NODELAY can have a significant negative impact of network performance. TCP_NODELAY is the only supported socket option
which uses *level* IPPROTO_TCP; all other options use level
SOL_SOCKET.
The following options are supported for **setsockopt()**. The Type identifies the type of data addressed by *optval*.

Value	Type	Meaning
SO_BROADCAST	BOOL	Allow transmission of broadcast messages on the socket.
SO_DEBUG	BOOL	Record debugging information.
SO_DONTLINGER	BOOL	Don't block close waiting for unsent data to be sent. Setting this option is equivalent to setting SO_LINGER with *l_onoff* set to zero.
SO_DONTROUTE	BOOL	Don't route: send directly to interface.
SO_KEEPALIVE	BOOL	Send keepalives
SO_LINGER	struct linger *	Linger on close if unsent data is present
SO_OOBINLINE	BOOL	Receive out-of-band data in the normal data stream.
SO_RCVBUF	Int	Specify buffer size for receives
SO_REUSEADDR	BOOL	Allow the socket to be bound to an address which is already in use. (See **bind()**.)

SO_SNDBUF	Int	Specify buffer size for sends.
TCP_NODELAY	BOOL	Disables the Nagle algorithm for send coalescing.

BSD options not supported for **setsockopt()** are:

Value	Type	Meaning
SO_ACCEPTCONN	BOOL	Socket is listening
SO_ERROR	Int	Get error status and clear
SO_RCVLOWAT	Int	Receive low water mark
SO_RCVTIMEO	Int	Receive timeout
SO_SNDLOWAT	Int	Send low water mark
SO_SNDTIMEO	Int	Send timeout
SO_TYPE	Int	Type of the socket
IP_OPTIONS		Set options field in IP header.

See Also

bind(), getsockopt(), ioctlsocket(), socket().

shutdown

Syntax:
int shutdown (SOCKET *so*, int *how*)

Description:
Disables sends and/or receives on a socket.

Parameters	**Description**
so	A descriptor identifying a socket.
how	A flag that describes what types of operation will no longer be allowed.

Return Value
If no error occurs, **shutdown()** returns 0. Otherwise, a value of
SOCKET_ERROR is returned, and a specific error code is returned in errno**.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EINVAL	how is not valid.
ENOTCONN	The socket is not connected (SOCK_STREAM only).
ENOTSOCK	The descriptor is not a socket.

Remarks

shutdown() is used on all types of sockets to disable reception, transmission, or both. If *how* is 0, subsequent receives on the socket will be disallowed. This has no effect on the lower protocol layers. For TCP, the TCP window is not changed and incoming data will be accepted (but not acknowledged) until the window is exhausted. For UDP, incoming datagrams are accepted and queued. In no case will an ICMP error packet be generated.

If *how* is 1, subsequent sends are disallowed. For TCP sockets, a FIN will be sent. Setting *how* to 2 disables both sends and receives as described above.
Note that **shutdown()** does not close the socket, and resources attached to the socket will not be freed until **closesocket()** is invoked.

Comments

shutdown() does not block regardless of the SO_LINGER setting on the socket. An application should not re-use a socket after it has been shut down.

See Also

connect(), socket().

socket

Syntax:
SOCKET socket (int *af*, int *type*, int *protocol*)

Description:
Creates a socket.

Parameters

af	An address format specification. The only format currently supported is PF_INET, which is the ARPA Internet address format.
type	A type specification for the new socket.
protocol	A particular protocol to be used with the socket, or 0 if the caller does not wish to specify a protocol.

Return Value
If no error occurs, **socket()** returns a descriptor referencing the new socket. Otherwise, a value of INVALID_SOCKET is returned, and a specific error code is returned in errno**.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
EAFNOSUPPORT	The specified address family is not supported.
EMFILE	No more file descriptors are available.
ENOBUFS	No buffer space is available. The socket cannot be created.
EPROTONOSUPPORT	The specified protocol is not supported.
EPROTOTYPE	The specified protocol is the wrong type for this socket.
ESOCKTNOSUPPORT	The specified socket type is not supported in this address family.

Remarks
socket() allocates a socket descriptor of the specified address family, data type and protocol, as well as related resources. If a protocol is not specified (i.e. equal to 0), the default for the specified connection mode is used.

Only a single protocol exists to support a particular socket type using a given address format. However, the address family may be given as AF_UNSPEC (unspecified), in which case the *protocol* parameter must be specified. The protocol number to use is particular to the
"communication domain" in which communication is to take place.

The following *type* specifications are supported:

Type	**Explanation**
SOCK_STREAM	Provides sequenced, reliable, two-way, connection-based byte streams with an out- of-band data transmission mechanism. Uses TCP for the Internet address family.
SOCK_DGRAM	Supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. Uses UDP for the Internet address family.

Sockets of type SOCK_STREAM are full-duplex byte streams. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a **connect**() call. Once connected, data may be transferred using **send**() and **recv**() calls. When a session has been completed, a **closesocket**() must be performed. Out-of-band data may also be transmitted as described in **send**() and received as described in **recv**().

The communications protocols used to implement a SOCK_STREAM ensure that data is not lost or duplicated. If data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, the connection is considered broken and subsequent calls will fail with the error code set to ETIMEDOUT.

SOCK_DGRAM sockets allow sending and receiving of datagrams to and from arbitrary peers using **sendto()** and **recvfrom()**. If such a socket is **connect()**ed to a specific peer, datagrams may be send to that peer **send**() and may be received from (only) this peer using **recv**().

See Also

accept(), bind(), connect(), getsockname(), getsockopt(), setsockopt(), listen(), recv(), recvfrom(), select(), send(), sendto(),

shutdown(),
ioctlsocket().

---

gethostbyaddr

Syntax:
struct hostent * gethostbyaddr (const char * *pcAddr*, int *len*, int *type*)

Description:
Gets host information corresponding to an address.

Parameters	**Description**
pcAddr	A pointer to an address in network byte order.
len	The length of the address, which must be 4 for PF_INET addresses.
type	The type of the address, which must be PF_INET.

Return Value
If no error occurs, **gethostbyaddr()** returns a pointer to the hostent structure described above. Otherwise it returns a NULL pointer and a specific error number is returned in errno**.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
WSAHOST_NOT_FOUND	Authoritative Answer Host not found.
WSATRY_AGAIN	Non-Authoritative Host not found, or SERVERFAIL.
WSANO_RECOVERY	Non-recoverable errors, FORMERR, REFUSED, NOTIMP.
WSANO_DATA	Valid name, no data record of requested type.

Remarks
gethostbyaddr() returns a pointer to the following structure which contains the name(s) and address which correspond to the given address.

```
struct hostent
{
        char * h_name;
        char ** h_aliases;
        short h_addrtype;
```

```
        short h_length;
        char ** h_addr_list;
};
```

The members of this structure are:

Element	Usage
h_name	Official name of the host (PC).
h_aliases	A NULL-terminated array of alternate names.
h_addrtype	The type of address being returned; for SOCKETS this is always PF_INET.
h_length	The length, in bytes, of each address; for PF_INET, this is always 4.
h_addr_list	A NULL-terminated list of addresses for the host. Addresses are returned in network byte order.

The macro h_addr is defined to be h_addr_list[0] for compatibility with older software. The pointer which is returned points to a structure which is allocated by SOCKETS. The application must never attempt to modify this structure or to free any of its components. The application should copy any information which it needs before issuing any other SOCKETS API calls.

See Also
gethostbyname(),

gethostbyname

Syntax:
struct hostent * gethostbyname (const char * *pszName*)

Description:
Gets host information corresponding to a hostname.

Parameters	**Description**
PszName	A pointer to the name of the host.

Return Value
If no error occurs, **gethostbyname()** returns a pointer to the hostent structure described above. Otherwise it returns a NULL pointer and a specific error number is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
WSAHOST_NOT_FOUND	Authoritative Answer Host not found.
WSATRY_AGAIN	Non-Authoritative Host not found, or SERVERFAIL.
WSANO_RECOVERY	Non recoverable errors, FORMERR, REFUSED, NOTIMP.
WSANO_DATA	Valid name, no data record of requested type.

Remarks
gethostbyname() returns a pointer to a hostent structure as described under **gethostbyaddr**(). The contents of this structure correspond to the hostname *pszName*.
The pointer which is returned points to a structure which is allocated by SOCKETS. The application must never attempt to modify this structure or to free any of its components. The application should copy any information which it needs before issuing any other SOCKETS API calls.
A **gethostbyname()** implementation must not resolve IP address strings passed to it. Such a request should be treated exactly as if an unknown host name were passed. An application with an IP address string to resolve should use **inet_addr()** to convert the string to an IP address, then **gethostbyaddr()** to obtain the hostent structure.

See Also

gethostname

Syntax:
int gethostname (char * *pszName*, int *iAddressLen*)

Description:
Return the standard host name for the local machine.

Parameters	**Description**
pszName	A pointer to a buffer that will receive the host name.
iAddressLen	The length of the buffer.

Return Value
If no error occurs, **gethostname()** returns 0, otherwise it returns
SOCKET_ERROR and a specific error code is returned in errno.

Error Codes

EFAULT	The *iAddressLen* parameter is too small
ENETDOWN	SOCKETS has detected that the network subsystem has failed.

Remarks
This routine returns the name of the local host into the buffer specified by the *pszName* parameter. The host name is returned as a null-
terminated string. The form of the host name is dependent on the
SOCKETS configuration file. However, it is guaranteed that the name returned will be successfully parsed by **gethostbyname().**

See Also
gethostbyname().

getprotobyname

Syntax:
struct protoent * getprotobyname (const char * *pszName*)

Description:
Gets protocol information corresponding to a protocol name.

Parameters	**Description**
pszName	A pointer to a protocol name.

Return Value
If no error occurs, **getprotobyname()** returns a pointer to the protoent structure described above. Otherwise it returns a NULL pointer and a specific error number is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
WSANO_RECOVERY	Non recoverable errors, FORMERR, REFUSED, NOTIMP.
WSANO_DATA	Valid name, no data record of requested type.

Remarks
getprotobyname() returns a pointer to the following structure which contains the name(s) and protocol number which correspond to the given protocol *pszName*.

```
struct protoent
{ char * p_name;
char ** p_aliases;
short p_proto;
};
```

The members of this structure are:

Element	**Usage**
p_name	Official name of the protocol.
p_aliases	A NULL-terminated array of alternate names.
p_proto	The protocol number, in host byte order.

The pointer which is returned points to a structure which is allocated by the SOCKETS library. The application must never attempt to modify this structure or to free any of its components. The application should copy any information which it needs before issuing any other
SOCKETS API calls.

See Also
getprotobynumber()

getprotobynumber

Syntax:
struct protoent * getprotobynumber (int *number*)

Description:
Gets protocol information corresponding to a protocol number.

Parameters	**Description**
number	A protocol number, in host byte order.

Return Value
If no error occurs, **getprotobynumber()** returns a pointer to the protoent structure described above. Otherwise it returns a NULL
pointer and a specific error number is returned in errno.

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
WSANO_RECOVERY	Non recoverable errors, FORMERR, REFUSED, NOTIMP.
WSANO_DATA	Valid name, no data record of requested type.

Remarks
This function returns a pointer to a protoent struct ure as described above in **getprotobyname()**. The contents of the structure correspond
to the given protocol number.

The pointer which is returned points to a structure which is allocated by SOCKETS. The application must never attempt to modify this struct ure or to free any of its components. The application should copy any information which it needs before issuing any other SOCKETS API calls.

See Also
getprotobyname()

getservbyname

Syntax:
struct servent * getservbyname (const char * *pszName*, const char * *proto*)

Description:
Gets service information corresponding to a service name and protocol.

Parameters	**Description**
pszName	A pointer to a service name.
proto	An optional pointer to a protocol name. If this is NULL, **getservbyname()** returns the first service entry for which the *pszName* matches the s_name or one of the s_aliases. Otherwise **getservbyname()** matches both the *pszName* and the *proto*.

Return Value
If no error occurs, **getservbyname()** returns a pointer to the servent structure described above. Otherwise it returns a NULL pointer and a specific error number is returned in errno**.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
WSANO_RECOVERY	Non recoverable errors, FORMERR, REFUSED, NOTIMP.
WSANO_DATA	Valid name, no data record of requested type.

Remarks
getservbyname() returns a pointer to the following structure which contains the name(s) and service number which correspond to the
given service *pszName*.

```
struct servent
{
        char * s_name;
        char ** s_aliases;
        short s_port;
        char * s_proto;
};
```

The members of this structure are:

Element	Usage
s_name	Official name of the service.
s_aliases	A NULL-terminated array of alternate names.
s_port	The port number at which the be contacted. Port numbers are returned in network byte order.
s_proto	The name of the protocol to use when contacting the service.

The pointer which is returned points to a structure which is allocated by the SOCKETS library. The application must never attempt to modify this structure or to free any of its components. The application should copy any information which it needs before issuing
any other SOCKETS API calls.

See Also
getservbyport()

getservbyport

Syntax
struct servent * getservbyport (int *port*, const char * *proto*)

Description:
Gets service information corresponding to a port and protocol.

Parameters	**Description**
Port	The port for a service, in network byte order.
Proto	An optional pointer to a protocol name. is NULL, **getservbyport()** returnsthe first service entry for which the *port* matches the s_port. Otherwise **getservbyport()** matches both the *port* and the *proto*.

Return Value
If no error occurs, **getservbyport()** returns a pointer to the servent structure described above. Otherwise it returns a NULL pointer and a specific error number is returned in errno**.**

Error Codes

ENETDOWN	SOCKETS has detected that the network subsystem has failed.
WSANO_RECOVERY	Non recoverable errors, FORMERR, REFUSED, NOTIMP.
WSANO_DATA	Valid name, no data record of requested type.

Remarks
getservbyport() returns a pointer a servent structure as described above for **getservbyname()**.

The pointer which is returned points to a structure which is allocated by SOCKETS. The application must never attempt to modify this structure or to free any of its components. The application should copy any information which it needs before issuing any other SOCKETS API calls.

See Also
getservbyname()

5.3.7 HTTP Functions (CGI_LIB*.LIB)

CGI Application API (Server API)

Introduction

The SOCKETS web servers, HTTPD.EXE and HTTPFTPD.EXE, provide both a Spawning Common Gateway Interface (CGI) and an Extention API with the ability to extend the server to create dynamic web pages, perform specialized tasks, etc. One of the extentions provided is a Server Side Includes (SSI) interface using the CGI interface, enabling a user to create web pages using HTML templates with variable names, which is substituted in-time with specific values The HTTPD Extension CGI works as follows: The extension has to implement one function called the *callback function*. The server has a number of functions that the extension may use, e.g. *HttpSendData*. They are designed to give the extension sufficient control over any http request.

Spawning CGI

An external program, indicated by the requested URL, is spawned. All relevant information is passed as environment variables. The program gets all input (e.g. posted data) from *standard in* and sends all response through *standard out*.

This type of CGI is discouraged in favor of the *Extension API*.

The following CGI environment variables are supported:

CONTENT_TYPE
CONTENT_LENGTH
PATH, COMSPEC
REQUEST_METHOD

Enough free memory must be available when spawning a CGI program, or no swapping or overlaying will be attempted. Since COMMAND.COM uses all free memory, it follows that no CGI program will be spawned if COMMAND.COM is the current foreground program.

CGI programs must be small and must execute reasonably quickly.
While a CGI program is executing, the HTTP server is effectively blocked and cannot service any other requests. No console input or output should be used. A CGI program is invoked by a URL containing
a path of /cgi-bin/<cgi-program> where <cgi-program> is the name of an executable program which must be in the HTTP root directory or in the path. Note that the "/cgi-bin/" part is stripped off and does not represent a real directory. <Cgi-program> may be followed by a "?" and a command line. On entry to the CGI program, the environment variables listed above are set up and can be accessed.

If a command line is given, it can also be accessed in the normal way. The CGI program generates a dynamic page by writing to STDOUT. When the CGI program terminates, this output is sent to the remote client (browser). The output can consist of a header and a body part separated by an empty line. If the header contains a “Content-type:” line, the content type will be set to that type and only the body will be sent to the client. Otherwise all the output will be sent to the client using content type “text/plain”. COMMAND.COM can be invoked as a CGI program to perform simple DOS functions e.g. directory listings. The following example performs a directory listing:

http://www.embedded-server.com/cgi-bin/command?/cdir

The next one performs a wide directory listing using a wild-card specification:

http://www.embedded-server.com/cgi-bin/command?/cdir%20*.htm%20/w

Note the use of %20 to specify a space character.

Refer to the INDEX.HTM web page for an example of various ways of calling CGI programs. The NUM.EXE program with source code NUM.C, demonstrates the use of a header and body part building a simple “page visited” web page:

```
printf("Content-type: text/html\n\n”
”<html>\n<h1>\nThis page has been visited %d times\n</h1>\n",
number);
printf("<P><P><A HREF=\"/index.htm\">Back</A>.</html>\n");
```

Forms programming can be performed using either the GET or POST methods. When GET is used, form data is copied to the command line and is limited to 128 characters including the URL part. When the POST method is used, the command line is also built. In addition,
form data are available from STDIN and is limited by disk space only.

See the forms programming example consisting of FORM.HTM, FORM.EXE and FORM.C for examples of using both the GET and POST methods. So that you may fully understand CGI programming, this detailed explanation of the server operation is provided.

Whenever HTTPD receives a URL containing “/cgi-bin/”, it interprets the rest of the URL as a DOS program to spawn and run to completion. The full path parsed from the URL is used, implying that the program should be in physical directory called “/cgi-bin/” or a subdirectory thereof. E.g. “program.exe” should be in “%HTTP_DIR%\cgi-bin\” if the request is “GET /cgibin/program.exe”.

While this “CGI program” is executing, the server can accept new server connections, but will not respond to them before the CGI program terminates. The CGI program can be any DOS program that
is small enough to fit into available memory. Since HTTPD is blocked while the CGI program executes, user interaction should not be used and the CGI program should complete in a reasonable time.
Operation on receiving a CGI URL:

If the CGI program name is followed by a "?", the rest of the line is sent as a command line to the CGI program after converting all %n combinations.

If a “Content-Type” header is encountered, the CONTENT_TYPE environment variable is set to the given value and if a “Content- Length” header is encountered, the CONTENT_LENGTH environment variable is set to the given value. The PATH and COMSPEC environment variables are copied to the new environment and the REQUEST_METHOD environment variable is set to either GET or POST.

If the POST method is used, the rest of the HTTP message is copied

to a temporary file that is then re-directed to stdin. The stdout stream is redirected to another temporary file. After completion of the request, the temporary files are deleted. They will be created in the %HTTPTMP% directory.

The CGI program is now invoked. This program can check the environment variables, access the command line and in the case of a POST, read from stdin.

All output that should be passed back to the HTTP client (Browser) is written to stdout. A single header line followed by an empty line, containing “Contenttype:

content_type” may be pre -pended to the data. This line will be used to set the content-type of the data being sent back. If such a header is not found, the content type will be set to “text/plain”.

Overview of the Extention API

The SOCKETS HTTP servers (HTTPD/HTTPFTPD) provide a facility to call functions in other modules which may be TSR or transient programs.

These functions are referred to as “HTTPD extensions”. HTTPD or HTTPFTPD must be loaded as a TSR using the /r switch. It provides an API via software Interrupt 63Hex. The API can be located by searching for a signature containing SockHTTPD starting 10 bytes before the interrupt entry point and terminated by a 0 byte.

A **CGI adapter** is provided that simplifies the communication with the server. It is located in a file called CGIADAP.C. The adapter finds the signature and provides a C interface. It also intercepts the callback function and performs a stack and context switch, which makes implementing an extension much easier.

An HTTPD extension registers interest in a specific URL by calling the **HttpRegister()** API specifying a “path”. Note that this path has nothing to do with an actual file path on the server and will override any real path that may be used for serving static pages. The **HttpRegister()** function also specifies a Callback function to be called when the actual request is received by HTTPD, a DWORD User ID to be used in callbacks and whether requests should be allowed to overlap, i.e. a new request can be received while still servicing a previous request or requests.

The Callback function will be called when a request for the registered path is received and as many times afterwards as is necessary to complete the request. It is called with a parameter structure specifying the reason for the request, the User ID, an HTTPD handle and values
specific to the reason for the callback, e.g. a pointer to the command line on the initial callback. Other reasons for calling the Callback function are to notify of new received data, connection closure by the peer, readiness to accept more data and connection errors.

The callback must return a value to indicate that it is still busy handling the request, has completed the request or wants to abort the request with an error. The HTTPD handle will be constant and unique from the first callback to the completion of the request.

While in the Callback function, data can be read from the peer or sent to the peer and a file can be submitted to be sent to the peer.

Note: Extensions are responsible for sending all HTTP header fields to clients.

The following extensions have been developed for functional and demonstrational purposes.

SSI Interface

If you want to display the current date and time, or a certain CGI environment variable in your otherwise static document, you can go through the trouble of writing a CGI program that outputs this small amount of virtual data. Or better yet, you can use a powerful feature called Server Side Includes (or SSI).

Server Side Includes are directives which you can place into your HTML documents to output such data as environment variables and file statistics. For a detailed introduction, please visit http://www.ora.com/info/cgi/ch05.html

A simple yet powerful interface is provided to perform Server Side Includes (SSI) tasks. A user only has to implement one predefined function and make use of only four API functions to unlock the power
of SSI. The working of the interface is described at the

top of the header file *ssi.h*.

To use, include *ssicgi.c* in your project and include *ssi.h* in your source files. Take a look at *ssi.c* for a simple example.

Extention API Examples

Five very simple examples are included to demonstrate the usage of the Extention API. Source code is included, as well as make files.
Put all *.htm* and *.exe* files in the %HTTP_DIR% directory and start *HTTPD*. Load all the cgi programs (you may use *cgi.bat*). All is in place now and the examples may be accessed through *index.htm*.

The first four examples may operate in one of two modes:

- ⌘ As a TSR (resident) program: this is the default behavior. At this stage unloading of the TSR is not supported. De-registration is possible by loading the program again. This routine may be repeated.

- ⌘ As a transient program: use '/t' command line switch to activate. This option will immediately spawn 'command.com'. From this pro mpt other cgi programs may be loaded. The program exits when 'command.com' is exited by typing 'exit' at the prompt.

These programs are:

1. ***cgiecho*** A very simple program that accepts data from a user and echoes it back nicely formatted. Get *echoform.htm* from the browser.
2. ***cgicount*** A page visit counter. Only updates between sessions if transient (cgicount /t). Get *num.htm* from the browser.
3. ***cgiform*** Does the same as the old 'fill out the form and submit'
utility. Get *caform.htm* from the browser.

SSI A very simple SSI implementation that demonstrates the SSI interfaces. *Template.htm* is filled by some variables. Get *ssi.htm* from the browser.

The fifth example, ***FFUR***, (Form-base File Upload Receiver) is only a transient program, but can easily be adapted to be similar to the rest. It handles the upload of a file as a POST command by filling out
ffur.htm.

HTTPD Function Reference

CGIADAP.C is an interface program a user may utilize to implement external extention CGI programs. This interface performs stack and context switches, and provides ordinary C functions to access the http server (*HTTPD.exe*).

The header file to use is CGIADAP.H.

The API may be used without using CGIADAP by making low level calls which are detailed below. In this case the user must perform the required stack and context switches if required.

HttpRegister

Syntax:
int HttpRegister(char far *szPath, int (far **pfCallback*) (HTTP_PARAMS far **psHttpParams*), int *iFlags*, DWORD *dwUserID*)

Description:
The HttpRegister() function registers an interest in a URL, providing a callback function. The callback is guaranteed to only be called when DOS can be called. The DOS critical handler will be disabled and all critical errors will result in an access error without any user
intervention. Since the callback happens at interrupt time, it should execute for as short a time as possible. After a done or error return,
no further callbacks will be generated for the current request.
Only one callback will be active at any time. Calling an API function while executing the callback function will not result in another callback before the current callback has returned.

Parameters	**Description**
szPath	Far pointer to the string identifying a URL. It should be an exact match of the abs_path part of the URI minus the leading '/'. For instance, If you want to capture all http://myserver.com/cgi-bin/getpage.exe, you should register 'cgibin/getpage.exe'.
pfCallback	Address of callback function. Return values when returning from callback: RET_OK not done, give me more upcalls RET_DONE done, no more upcalls please RET_ERR done, error
psHttpParams	Far pointer to HTTP parameter block.

psHttpParams->iReason
Reason for callback:

R_NEWREQ - New HTTP request. pszCommandLine points to the command line passed in the URL. The number contained in iValue specifies the HTTP operation; RQ_GET for GET and RQ_POST for POST.

R_INDATA - Input data available, iValue contains count.

R_OUTDATA - Can send output data, iValue contains count.
R_ENDDATA - Peer closed connection i.e. "end of input data"
R_CLOSED - Connection closed.

psHttpParams->iHandle
HTTPD handle, used in subsequent API calls for this request. The user should not modify it. See HTAPIC.H for the other definitions

iflags	F_OVERLAP - Overlapped request (1), non-overlapped request (0). All other bits are reserved.
dwUserID	Value passed to HttpRegister(); this value is for use by the extension, HTTPD does not modify it.

Return value

0	OK
< 0	One of the error messages (SEE HTAPIC.H)

Low level calling parameters

AH	APIF_REGISTER (0)
DS:SI	szPath
ES:DI	pfCallback
BX	iFlags
CX:DX	dwUserID

Low level return parameters
Return code in AX.

Note that the stack and the data segment on entry will be that of HTTPD. Depending on the memory model used for the extension and the amount of stack space required, it may be required to switch stacks during the callback.

HttpDeRegister

Syntax:
int HttpDeRegister(char far **pszPath*)

Description:
The HttpDeRegister() function removes the interest in a URL. After
this call no more callbacks will be generated for this URL. Any requests in progress will be terminated with an error to the peer. This function must be called for all registrations made by a program before terminating that program; otherwise the system will inevitably crash on any subsequent request.

Parameters	**Description**
pszPath	Far pointer to URL to de-register.

Return value

0	OK
< 0	One of the error messages (SEE HTAPIC.H)

Low level calling parameters

AH	APIF_DEREGISTER (1)
DS:SI	pszPath

Low level return parameters
Return code in AX.

HttpGetData

Syntax
int HttpGetData(int *iHandle*, char far **pcBuf*, int *iCount*)

Description:
The HttpGetData() function can be called when a POST operation has been indicated by the callback to get data sent to the server by the client. If more data is expected and the extension is busy executing the callback function, a 0 return should be made from the callback indicating it is still busy and getting more data should be attempted at the next callback.

Parameter	**Description**
iHandle	Handle passed in pfsHttpParams.
pcBuf	Far pointer to buffer to receive data.
iCount	Length of buffer.

Return value

>=0	OK, number of bytes received.
< 0	One of the error messages (SEE HTAPIC.H)

Low level calling parameters

AH	APIF_GETDATA (2)
BX	iHandle
DS:SI	pcBuf
CX	iCount

Low level return parameters
Return code in AX.

HttpSendData

Syntax
int HttpSendData(int *iHandle*, char far **pcBuf*, int *iCount*)

Description:
The HttpSendData() function is used to send data to the client. If the return indicates that less than the requested number of bytes has
been sent and the extension is busy executing the callback function, a 0 return should be made from the callback indicating it is still busy. Then an attempt to send more data should be made at the next callback. All the required data should be sent to the client before an HttpSubmitFile() function is used. After HttpSubmitFile(), HttpSendData() should not be called again.

Parameter	**Description**
iHandle	Handle passed in pfsHttpParams.
pcBuf	Far pointer to buffer with data to send.
iCount	Length of buffer.

Return value

>=0	Number of bytes actually sent
< 0	One of the error messages (SEE HTAPIC.H)

Low level calling parameters

AH	APIF_SENDDATA (3)
BX	iHandle
DS:SI	pcBuf
CX	iCount

Low level return parameters
Return code in AX.

HttpSubmitFile

Syntax:
int HttpSubmitFile(int *iHandle*, char far **pszFileName*)

Description:
The HttpSubmitFile() function is used to submit a file to be sent to the client in response to a request. The file will be logically appended to any data already sent using HttpSendData(). The file should not be exclusively opened when it is submitted. After it is transmitted, transmit upcalls will be issued normally. This gives the user the ability to send any number of files on the connection with arbitrary data in between.

Parameter	**Description**
iHandle	Handle passed in pfsHttpParams.
pszFileName	Far pointer to name of file to submit.

Return value

0	OK
< 0	One of the error messages (SEE HTAPIC.H)

Low level calling parameters

AH	APIF_SENDFILE (4)
BX	iHandle
DS:SI	pszFileName

HttpGetStatus

Syntax
int HttpGetStatus(void)

Description:
The HttpGetStatus() function gets the number of connections to the server. It must also be used as a polling function when the server is
running in passive mode to dequeue and handle pending requests.

Parameters
None.

Return value

>=0	Number of connections to server.
< 0	One of the error messages (SEE HTAPIC.H)

Low level calling parameters

AH	APIF_GETSTATUS(6)

Low level return parameters
Return code in AX.

HttpGetVersion

Syntax
int **HttpGetVersion**(void);

Description:
The **HttpGetVersion()** function gets the version of the running HTTP server.

Parameter
None.

Return value

>=0	Version number.
< 0	One of the error messages (SEE HTAPIC.H)

Low level calling parameters

AH	APIF_GETVERSION (5)

Low level return parameters
Return code in AX.

GetStackPointer
GetStackSegment

Syntax
int GetStackPointer (void)
int GetStackSegment (void)

Description:
The GetStackPointer and GetStackSegment functions get the current Stack Pointer/Segment.

Parameters
None.

Return value
Current value of Stack Pointer/Segment.

SetStackPointer
SetStackSegment

Syntax:
void SetStackPointer (int iPointer)
void SetStackSegment (int iSegment)

Description:
The SetStackPointer and SetStackSegment functions set the Stack
Pointer/Segment. The stack pointer for callbacks is by default set to _SP - 1000, the first time the HTTP API is called. If you would need space on the stack, or for some reason want to make it tighter, set the stack pointer for callbacks manually. Be careful not to overwrite used memory.

Parameters	**Description**
iPointer	Value to set Stack Pointer to.

Return value
None.

Constants and Definitions used by CGI API
Refer to HTAPIC.H.

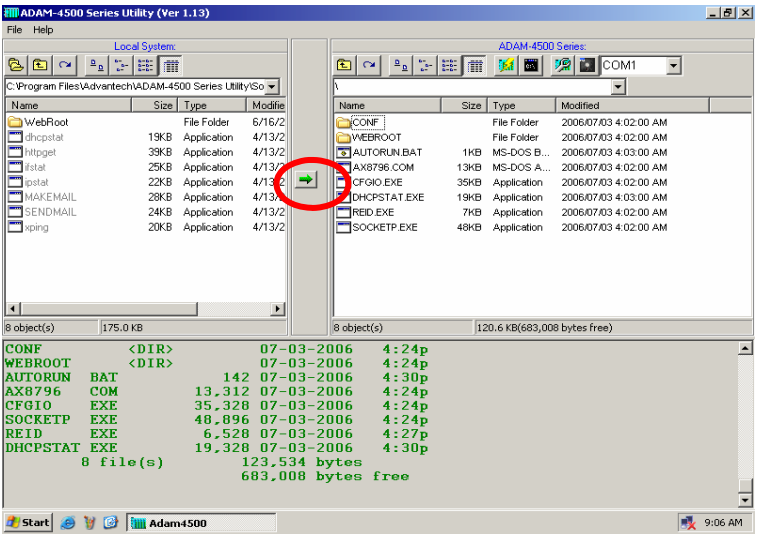
SSI Definitions and functions
Refer to SSI.H.

6

Sockets Utility

Chapter 6 Sockets Utility

SOCKETS utilities make use of command-line parameters and/or
configuration files. Please be careful to note the name and location of the configuration file used by the application you are working with. All SOCKETS applications require that the kernel be loaded before the
application is run in order to function properly. You can find all these SOCKETS utilities under **C:\Program Files\Advantech\ADAM-4500
Series Utility\Source\Drive_D\Extension_files** (Refer to image below). When you need to use the utilities, remember to download the execution file to ADAM-4500 Series Controller.


ADAM-4500 Series Utility (Ver 1.13)
File Help
Local System:
ADAM-4500 Series:
C:\Program Files\Advantech\ADAM-4500 Series Utility\So
COM1
Name Size Type Modifie
WebRoot File Folder 6/16/2
dhcpstat 19KB Application 4/13/2
httpget 39KB Application 4/13/2
ifstat 25KB Application 4/13/2
ipstat 22KB Application 4/13/2
MAKEMAIL 28KB Application 4/13/2
SENDMAIL 24KB Application 4/13/2
xping 20KB Application 4/13/2
Name Size Type Modified
CONF File Folder 2006/07/03 4:02:00 AM
WEBROOT File Folder 2006/07/03 4:02:00 AM
AUTORUN.BAT 1KB MS-DOS B... 2006/07/03 4:03:00 AM
AX8796.COM 13KB MS-DOS A... 2006/07/03 4:02:00 AM
CFGIO.EXE 35KB Application 2006/07/03 4:02:00 AM
DHCPSTAT.EXE 19KB Application 2006/07/03 4:03:00 AM
REID.EXE 7KB Application 2006/07/03 4:02:00 AM
SOCKETP.EXE 48KB Application 2006/07/03 4:02:00 AM
8 object(s) 175.0 KB
8 object(s) 120.6 KB(683,008 bytes free)
CONF <DIR> 07-03-2006 4:24p
WEBROOT <DIR> 07-03-2006 4:24p
AUTORUN BAT 142 07-03-2006 4:30p
AX8796 COM 13,312 07-03-2006 4:24p
CFGIO EXE 35,328 07-03-2006 4:24p
SOCKETP EXE 48,896 07-03-2006 4:24p
REID EXE 6,528 07-03-2006 4:27p
DHCPSTAT EXE 19,328 07-03-2006 4:30p
8 file(s) 123,534 bytes
683,008 bytes free
Start
Adam4500
9:06 AM

DHCPSTAT

DHCPSTAT displays the DHCP information for the machine.

Syntax
DHCPSTAT [r | v]

Options
r force a renewal of the DHCP lease.
V display the SOCKETS version information.

Example

DHCPSTAT
This will display all the DHCP information, such as IP address and lease time.

FTP

FTP is a file transmitting and retrieving client that runs in interactive or batch mode.

Syntax
FTP server [options]

Options
/n
/v
/p=Port
/f=ScriptFile [ScriptParameters]

Remarks
Server
The name or IP address of a server you want to connect.
/n
Suppress progress indicator.
/v
Verbose output for troubleshooting.
/p=Port
Connect to a server port other than the standard FTP port number of 21.
/f=ScriptFile
A file containing commands for the client to send to the server upon connection. Simple parameter substitution is performed, with the first element
of *ScriptParameters* accessible as “%1,” etc.
ScriptParameters
Parameters to pass into the *ScriptFile*.

Return Codes

0 Success
1 Parameter error
2 SOCKETS not loaded
3 User aborted
4 Transfer aborted
5 Error writing local file
6 Error reading local file

Other Server returned error response code; to find that error code, add 390 to the response code returned by FTP. The result will always be greater than or equal to 400 in this case.

Example

FTP /n FTP.cdrom.com /f=getfile.scr /.2/simtelnet/msdos DIRS.TXT
(The file GETFILE.SCR):
user anonymous
pass root@
cd %1
binary
get %2
quit

FTP Commands

The commands entered at the FTP client can be interpreted and translated to standard FTP commands to be sent to the server. The FTP server might recognize more, or less, commands than the standard list of commands as specified in RFC 959. The **site** command is always server dependent. Some of the standard commands are implemented differently in various servers. Useful things to note are:

1. The **put** and **get** commands allow multiple file transfers by usage of wild card characters. When **get**ting files with paths or long names, no translation of foreign file names is done. Specify a valid DOS ***local_file*** name.
2. A short directory list (NLST) is obtained by **ls** and the long list with **dir**.
3. Some of the commands can be abbreviated.
4. Some commands are aliases added for user comfort like **bye**, **exit** and **quit**; **get** and **mget**; and **put** and **mput**.
5. The optional **[*local_file*]** parameter will, when specified, cause the output of that command to be logged to a file. By specifying the file as PRN you can get immediate printouts.
6. On some servers you might specify the optional **[*remote_file*]** parameter as PRN or the printer output device to do remote printing. (See also the **site nopath** command for the SOCKETS FTP server.)
7. The **F3** key and **spacebar** can be used to recall the last command word by word. Below is a list of commands recognized by the SOCKETS FTP client (some FTP servers might not offer all the facilities):

Command	Description
abort	Cancel an incomplete transfer
append	"Put" a file at the server but append it if the file exists
ascii	Synonym for type a
binary	Synonym for type i
bye	Synonym for quit
cd directory	Synonym for cwd
cwd directory	Change server directory
dele file	Delete a server file
dir [file l directory [local_file]]	Synonym for list
exit	Synonym for quit
get remote_file(s) [local_file]	Transfer a file from the server in the current mode (type)
image	Synonym for type i
ls [file l directory [local_file]]	Synonym for nlst
lcd directory	Perform a local change directory
ldir [file l directory]	Give a local directory listing
list [file l directory [local_file]]	Give a long directory listing
mget remote_file(s) [local_file]	Synonym for get
mkdir remote_directory	Create a server directory
mput local_file(s) [remote_file]	Synonym for put
nlst [file l directory [local_file]]	Give a short names-only directory listing
pass [password]	Password for username
pasv [on | off]	Report or change the status of the passive transfer mode to enable firewall friendly file transfers. (The SOCKETS FTP client always tries to switch passive mode on at the start of a session.)
put local_file(s) [remote_file]	Transfer a file to the server in the current mode (type)
Pwd	Print working directory at server
quit	Terminate FTP session
quote remote_command [args ...]	Send a command to the server without any interpretation
rmdir remote_directory	Remove (delete) a server directory
rnfr existing_filename	Rename a file, command 1 of 2
rnto new_filename	Rename a file, command 2 of 2
site sub-command	Send server specific commands
size file	Report the file size in bytes as a 213 message
shell	Shell to DOS for IFTP.EXE
stat	Report the status of a transfer or active connections
System	Return operating system information from the server
type [i l a]	Report or select the file transfer mode: image (binary) or ASCII
user [username]	Username to logon
verbose [on | off]	Verbose mode reports more of the FTP negotiations

IFSTAT

IFSTAT displays the status of the Interface and the version information for SOCKETS.

Syntax

IFSTAT [i] [v]

Options

i show the Interface status.
v show the version information.

Example

IFSTAT v
This will display the SOCKETS version information

IPSTAT

The IPSTAT utility returns statistics on IP and memory. Use IPSTAT to check for error conditions and memory problems.

Syntax
IPSTAT

Example
IPSTAT

The following will be displayed (The values may differ):

IP stats at 160F:04C8:

Total Packets	2671
Smaller than minimum size	0
IP header length too small	0
Wrong IP version	0
Unsupported protocol	0
Memory available	9016
Memory allocation failures	0
Memory free errors	0
Minimum stack observed	886

MAKEMAIL

MAKEMAIL packages the body text and any attachments for delivery using the **SENDMAIL** application.

Syntax
MAKEMAIL –tToAddress –fFromAddress –sSubject –bBodyTextFile -oOutputFileaAttachment

Options
ToAddress
The e-mail address of the recipient(s) of this mail. Additional recipients are specified by repeated use of the **–t** parameter. If the *ToAddress* is a name that can be resolved by either the DNS server or host file then the *@servername* is not necessary.
FromAddress
Used to identify the sender of the message
Subject
The subject line of the e-mail message
BodyTextFile
The local file containing the body text of the e-mail message to deliver
OutputFile
The local file name in which to store the prepared file for delivery by **SENDMAIL**.This file is overwritten if it already exists!
Attachment
The name of a local file to be binary attached to this e-mail message. Multiple attachments are created by repeated use of the **–a** parameter. Files are attached as MIME parts, encoded with the application/x- uuencode content type.

Example

MAKEMAIL tfred@yahoo.com fmary@yahoo.com –sStatus –bmessage.txt –omail.dat

MAKEMAIL –tfred –tbarney –fwilma –sDinner –bmenu.txt –omail.dat

MAKEMAIL –tfred –fwilma –sBowling –bbody.txt –aStone.jpg –
aRock.jpg –omail.dat

SENDMAIL

SENDMAIL delivers e-mail messages packaged by the **MAKEMAIL** application to an Internet mail server. **SENDMAIL** also creates a local log file to indicate successful send or failures.

Syntax
SENDMAIL server file

Options
Server
The IP address or DNS name of the Internet mail server to receive the message.
File
The file created by the **MAKEMAIL** utility to deliver.

Logging Format

Timestamp, Code String

Timestamp
Weekday Month Day Time Year

Code
Three digit integer: 000 means perfect success, 100-199 mean usage error and 200-299 means TCP/IP error from server.

String
Human–readable explanation of the error code

Example

SENDMAIL mail.datalight.com mail.dat

XPING

XPING starts a continuous string of pings until stopped by a keystroke.

Syntax
XPING *ip address* [interval]

Remarks
ip address This may be a numeric address or a name address.

Options
interval The time to wait between pings in clock ticks.

Example

XPING 10.0.0.1 20
This will ping the address of 10.0.0.1 every 20 clock ticks.

7

HTTP and FTP Server Application

HTTP Server

Overview

The SOCKETS HTTP server, HTTPD.EXE, is a small, fast, reliable and extendable web server that can run as either an application or TSR. Apart from the minimum required file download capability, the following additional capabilities are provided:

1. Remote Console Server -- ability to gain terminal-type access to the server system, using a standard browser, without the need to install any software on the browser computer.
2. Authentication – Both system wide and directory wise
3. CGI Extendibility – The ability to extend the server to create dynamic web pages, perform specialized tasks, etc.
4. A Server Side Includes (SSI) interface is provided using the CGI interface, enabling a user to create web pages using HTML templates with variable names substituted in-time with specific values.
5. Ability to run as a background process.
6. Flexibility to control physical parameters such as memory usage and number of connections.

Server

The HTTP server is used to send static web pages existing as files on the server or dynamically generated web pages to a remote client (browser). Dynamic pages can be generated in two ways:

1. **Extension CGI.** By calling an external CGI handler, the server provides an API to external handlers. A Server Side Includes (SSI) interface is provided as well, which makes it very easy to create powerful interactive web pages.
2. **Spawning CGI.** By spawning programs with a relatively short execution time to generate the pages through a mechanism similar to CGI, the basic mechanism used by CGI is that arbitrary programs can be spawned from the web server with input as received from the remote browser and output that can be sent to the browser.

The Remote Console Server accepts input from a remote client that is fed to the keyboard buffer for use by an arbitrary program using it. It also monitors the screen display buffer area and sends screen information to the remote client.

The SOCKETS password file controls authentication. Authentication is user specific and may also differ from directory to directory. It may also be put off for either some or all users. See the section on authentication.

The HTTP server can support multiple simultaneous sessions. The GET and POST request methods are implemented as well as the following MIME types: text/html, text/plain, image/gif, image/jpeg, image/jpeg and application/octet-stream. The MIME type is determined by the file extension.

Remote Console Server

Initialization

The client (browser) will initialize a remote session. An HTTP connection will be made to the HTTP server. The downloaded page will contain the applet that will automatically connect to the RCS on TCP port 81. An example download page is supplied as REMCON.HTM.

Almost any application e.g. a text editor can be run on the server. The remote keyboard and display control the application as if they were locally attached.

On the remote side, the Java Applet acts as a simple terminal emulator that displays what it receives from the server and sends what is entered from the keyboard to the server.

It is not required to have a real display adapter on the embedded system server, only to have display buffer memory.

When a new connection is made, all the screen data, as well as the cursor position, is sent to the client. Subsequently the RCS keeps a watch on the video memory and cursor position and whenever a change is detected, the RCS sends the changed data to the Java applet.

Keyboard data received from the client is passed to the keyboard buffer making it available as keyboard input for use by any application executing on the server.

Remote Console Client

The remote console client exists as a Java 1.3.1 applet, supplied as RC.JAR, and will function on any Java 1.3.1 compliant browser. Please note that a security certificate has not been compiled into RC.JAR so it is not compliant with versions of the Netscape browser that require a security certificate to run Java applets. A DOS based client using SOCKETS is also supplied as RCCLI.EXE. For additional information about RC.JAR or RCCLI.EXE, please see the Utility Description Chapter.

Extension CGI

The SOCKETS HTTP servers (HTTPD/HTTPFTPD) provide a facility to call functions in other modules which may be TSR or transient programs. These functions are referred to as "HTTPD extensions." For more information please see the "ROM - DOS Developer's Guide" section "CGI Application API.

Extension CGI Examples

Five very simple examples are included to demonstrate the implementation of CGI. Source code is included.

Put all *.htm* and *.exe* files in the %HTTP_DIR% directory and start *HTTPD*. Load all the cgi programs (you may use *cgi.bat*). All is in place now and the examples may be accessed through *index.htm*.

The first four examples may operate in one of two modes:

As a TSR (resident) program: this is the default behavior. At this stage unloading of the TSR is not supported. De -registration is possible by loading the program again. This routine may be repeated.

As a transient program: use '/t' command line switch to activate. This option will immediately spawn 'command.com'. From this prompt other cgi programs may be loaded. The program exits when 'command.com' is exited by typing 'exit' at the prompt.

These programs are:

1. ***cgiecho*** A very simple program that accepts data from a user and echoes it back nicely formatted. Get *echoform.htm* from the browser.

2. ***cgicount*** A page visit counter. Only updates between sessions if transient (cgicount /t). Get *num.htm* from the browser.

3. ***cgiform*** Does the same as the old 'fill out the form and submit' utility. Get *caform.htm* from the browser.

4. ***SSI*** A very simple SSI implementation demonstrating the SSI interfaces. *Template.htm* is filled by some variables. Get *ssi.htm* from the browser.

The fifth example, ***FFUR***, (Form-base File Upload Receiver) is only a transient program, but can easily be adapted to be similar to the rest. It handles the upload of a file as a POST command by filling out
ffur.htm.

Passive Mode

The server may be run in passive mode by specifying a '/p' command line switch. When passive, the server will record network events but only handle them once it is triggered by a CGI user.

Server Memory

The server's memory usage may be controlled in two ways:

1. By specifying the amount of memory when going TSR.

2. By specifying the maximum number of connections the server will allow.

Option 1 is the recommended option. Use Option 2 if you have
'heavy' web pages – usually the type where pages consist of frames and many images, etc. Connections are generally reset when more connections are attempted than the defined maximum. The client then must retry to establish the lost connections, leading to a more distributed load on the server.

Spawning CGI

An external program, indicated by the requested URL, is spawned. All relevant information is passed as environment variables. The CGI program gets all input (e.g. posted data) from *standard in* and sends all response through *standard out*. Spawning CGI is discouraged in favor of Extension *CGI*. For more information please see the "ROM - DOS Developer's Guide" section "CGI Application API.

Authentication

Default authentication matches the capabilities of the FTP server as documented in the section “FTP Server” on page 217. A file called "SOCKET.UPW" should exist in the SOCKETS (environment variable) directory.

The default permission file controls remote console access. Each listed user has a single -letter privilege code set if he has privilege to use the Remote Console. The code should be missing if that user does not have Remote Console privilege.

An additional authentication feature is implemented - **htaccess**. This feature provides a per-directory permission override mechanism. It is enabled using '/t' as command line switch. If htaccess is enabled, the default mechanism may be skipped (but no default users or remote console access will be available).

A file called HTACCESS (typically hidden) contains authentication overrides to enable partial anonymous access or additional password security to subdirectories, etc. If this feature is activated, the server code will look for HTACCESS files in each directory starting from the requested path and continuing upward in the directory structure (assuming the root directory to be at the top) until an HTACCESS file
is found. If no file is found, then the default settings are used. An anonymous access entry is available for the developer to specify that some subdirectory is authorized for any user, although its parent directory is password -protected. CGI scripts can also be controlled via the HTACCESS mechanism.

HTTPD Program

The syntax for HTTPD is:

HTTPD [options] [<http_port>] [<rc_port>]

Any combination of these switches may be used. They should be separated by at least one space.

Option	Description
/? /h	Display help screen
/r	Run server in TSR mode
/s	Display server status
/t	Enable htaccess directory level authentication
/u	Unload if resident
/c	Close listen
/d	Do not start remote console
/g	allow old type (spawning) CGI
/p	Passive mode
/i=<InterruptNumber>	Interrupt number for cgi API
/m=<MemorySize>	Set memory size
/n=<MaximumConnections>	Number of simultaneous connections
/a=<ScreenX>, <ScreenY>	Set screen aspect
/v=<ScreenBufferSegment>[:<ScreenBufferOffset>]	Set video buffer address (hex)
/k	Unload and abort all active connections

Remarks

ScreenX, ScreenY

The width and height of the screen area to serve for the remote console session. These values default to 80 and 25, respectively.

ScreenBufferSegment, ScreenBufferOffset

Together, a pointer to the top -left corner of the display memory to serve for the remote console session. These values default to B000 and 0000 respectively, for monochrome display adapters and to B800 and 0000 respectively, for color display adapters.

MemorySize

The maximum amount of memory available to the server. The default value is 32K. The value of m can range from 8192 to 63472.

MaximumConnections

The maximum number of simultaneous connections allowed by the server.

InterruptNumber
The interrupt number to access the CGI API.

http_port
HTTP port to listen on. This parameter defaults to the standard HTTP port number of 80.

rc_port
Remote Console port to listen on. This parameter defaults to 81.

The "root" directory for web content is the current directory when HTTPD is started. This can be changed by setting an environment variable HTTP_DIR e.g.

SET HTTP_DIR=D:\SERVER\WEB

Format of "SOCKET.UPW"

This is the same file used for the HTTP and FTP server's (*FTPD.EXE*) permissions. This file consists of lines where each line contains a user's information. A line starting with a # is considered a comment and is ignored. Each line consists of four fields:

<Username> <Password> <Working Directory> <Permissions> [# comment]

- Username: The name of this user. If it is *, it will be used when the client does not specify a username.

- Password: This user's password. If it is *, no password is required

- Working Directory: The user will only have access to this directory and its subdirectories. If it is '/', this user has access to the whole system. HTTP_DIR can be referred to as '\'. If a relative path is specified, it is appended to HTTP_DIR.

- Permissions: When a user is granted both FTP and HTTP permissions, the FTP permissions must appear **first**, otherwise they will be ignored. Permission are listed below:

  FTP Rights:
  - **d** change directories
  - **c** create/delete directories
  - **w** write files

r read files

HTTP Rights:

e get files
p post files
g use CGI
m use remote console

Fields should be separated by single spaces. If any field is missing the entry is ignored. A comment may follow the last field (permissions) of the line.

Format of "htaccess"

Any directory may contain this file, and serve as overrides to the general permissions for the containing directory and all its subs until another htaccess is found. This file consists of lines where each line contains a user's information. A line starting with a # is considered a comment and is ignored. Each line consists of three fields:

<Username> <Password> <Permissions> [# comment]

⌘ Username: The name of this user. If it is *, it will be used when the client does not specify a username.

⌘ Password: This user's password. If it is *, no password is required

⌘ Permissions: Operations allowed. Permission are listed below:

e - User may 'g**e**t' files
p - User may '**p**ost' files
g - User may use c**g**i

Fields should be separated by single spaces. If any field is missing the entry is ignored. A comment may follow the last field (permissions)
of the line.

Note: If a default user is supplied, it should always appear first in the list of users. Only users below the default user will be considered.

FTP Server

FTPD is a file server that can run either as an application or as a TSR. The name of the server as displayed in the banner is determined by the HOSTNAME environment variable. If the environment variable is not set, the name "Socket" is used. The user password file, SOCKET.UPW, in the SOCKETS directory (indicated by the SOCKETS environment variable) controls access.

A temporary file is created when a directory listing is requested. This file is created in the current directory, but can be created in any directory as specified in the FTPDIR environment variable.

FTPD Program

The syntax for FTPD is:

FTPD [options] [<ftp_port>]

Option	Description
/? /h	Display help screen
/r	Run server in TSR mode
/s	Display server status
/u	Unload if resident
/c	Close listen
/m=<MemorySize>	Set memory size
/n=<MaximumConnections>	Number of simultaneous connections
/k	Abort all active connections and unload

Remarks

MemorySize

The number of bytes of memory available to the server. This value defaults to 32768.

MaximumConnections

The maximum number of simultaneous connections allowed by the server.

ftp_port

FTPD will listen on the listed port. This parameter defaults to the standard FTP port number of 21.

Configuration File

FTPD uses the standard SOCKET.UPW file for validating logins. The file is composed of text lines, each representing a login name, password, and the configuration to use for a session opened with those credentials. Space characters separate the parameters in the file, which are in the following format:

name password directory rights

The location of the username/password file to be used by the server is specified by the environment variable SOCKETS as follows:

%SOCKETS%\SOCKET.UPW

If the variable SOCKETS is not specified, the following file is used:

\DL\SOCKETS\SOCKET.UPW

Configuration File Parameters

name
The login name of this record.

password
The password to authenticate a user trying to login as this name.

directory
The starting directory for this user.

rights
Up to four characters specifying which permissions this user is granted:

r means that this user has read access.

w means that this user has write access.

c means that this user has permission to make new directories.

d means that this user has permission to change to a directory other than his starting location and subdirectories from the starting location.

Example Socket.upw
Admin admin c:\ drwc
Guest * c:\guest dr

Example Command Line
FTPD /m=40000 /r

FTP Server Commands

The following commands are recognised by the SOCKETS FTP server:

Command	Description
Abort	cancel an incomplete transfer
append	"put" a file at the server but append it if the file exists
cwd *directory*	change server directory
dele *file*	delete a server file
list [*file* l *directory*]	give a long directory listing
mkd *remote_directory*	create a server directory
nlst [*file* l *directory*]	gives a short names-only directory listing
pass [*password*]	password for username
pasv [on | off]	report or change the status of the passive transfer mode to enable firewall friendly file transfers. (The SOCKETS FTP client always tries to switch passive mode on at the start of a session.)
retr *remote_file*	transfer a file from the server in the current mode
stor *local_file*	transfer a file to the server in the current mode
pwd	print working directory
quit	terminate FTP session
rmd *remote_directory*	remove (delete) directory
rnfr *existing_filename*	rename a file, command 1 of
2 **rnto *new_filename***	rename a file, command 2 of
2 **site [path l nopath]**	use full path description
site raw [*interface*]	open a session to a raw host using one of the raw lines (interfaces) specified
site *sub-command*	command to be passed on to raw host
size *file*	report the file size in bytes as a message prefixed with 213
stat	report the status of a transfer or active connections
system	return operating system information from the server
type [i l a]	report or select the file transfer mode image (binary) or ASCII
user [*username*]	username to logon

Combined HTTP and FTP Server

HTTPFTPD is a combined HTTP and FTP server that can run either as an application or as a TSR. By default, it processes normal HTTP requests on port 80 and normal FTP requests on port 21. It also serves a proprietary session displaying the contents of text -mode display memory to the **RC.JAR** and **RCCLI** client applications. This feature is commonly called the "remote console." If the **HTTPFTPD** server is loaded as a DOS TSR program, set the environment variable, HTTP_DIR, to the location of the INDEX.HTML file; for example, SET HTTP_DIR=C:\DL\SOCKETS\SERVER

HTTPFTPD Program

The syntax for FTTPD is:

HTTPFTPD [options] [<http_port> [<ftp_port> [<rc_port>]]]

Any combination of these switches may be used. They should be separated by at least one space.

Option	Description
/? /h	Display help screen
/r	Run server in TSR mode
/s	Display server status
/t	Enable htaccess directory level authentication
/u	Unload if resident
/c	Close listen
/d	Do not start remote console
/g	Allow old type (spawning) CGI
/p	Passive mode
/i=<InterruptNumber>	Interrupt number for cgi API
/m=<MemorySize>	set memory size
/n=<MaximumConnections>	number of simultaneous connections
/a=<ScreenX>, <ScreenY>	set screen aspec
/v=<ScreenBufferSegment>[:<ScreenBufferOffset>]	set video buffer address (hex)
/k	Abort all active connections and unload

Remarks

ScreenX, ScreenY

The width and height of the screen area to serve for the remote console session. These values default to 80 and 25, respectively.

ScreenBufferSegment, ScreenBufferOffset

Together, a pointer to the top-left corner of the display memory to serve for the remote console session. These values default to B000 and 0000 respectively, for monochrome display adapters and to B800 and 0000 respectively, for color display adapters.

MemorySize

The maximum amount of memory available to the server. The default value is 32K. The value of m can range from 8192 to 63472.

MaximumConnections

The maximum number of simultaneous connections allowed by the server.

InterruptNumber

The interrupt number to access the CGI API.

http_port

HTTP port to listen on. This parameter defaults to the standard HTTP port number of 80.

ftp_port

FTP port to listen on. This parameter defaults to the standard FTP port number of 21

rc_port

Remote Console port to listen on. This parameter defaults to 81.

Configuration File

HTTPFTPD uses the standard SOCKET.UPW file for validating logins. The file is composed of text lines, each representing a login name, password, and the configuration to use for a session opened with those credentials.

Space characters separate the parameters in the file, which are in the following format:

name password directory rights

The location of the username/password file to be used by the server is specified by the environment variable SOCKETS as follows:

%SOCKETS%\SOCKET.UPW

If the variable SOCKETS is not specified, the following file is used:

\DL\SOCKETS\SOCKET.UPW

Configuration File Parameters

name
The login name of this record.

password
The password to authenticate a user trying to login as this name.

directory
The starting directory for this user.

w means that this user has write access.

c means that this user has permission to make new directories.

d means that this user has permission to change to a directory other than his starting location and subdirectories from the starting location.

e means that this user may 'g**e**t' files

p means that this user may '**p**ost' files

g means that this user may use c**g**i

m means that this user may use Re**m**ote Console

Example Command Lines

HTTPFTPD /m=40000 /r
HTTPFTPD /a=80,25 /v=a000:0000 /r

Appendix A
COM Port Register Structure

Appendix A COM Port Register Structure

This appendix gives a short description of each module's registers.
For more information, please refer to the STARTECH 16C550 UART chip data book. All registers are one byte. Bit 0 is the least significant bit, and bit 7 is the most significant bit. The address of each register is specified as an offset from the port base address (BASE), COM1 is 3F8h and COM2 is 2F8h.

DLAB is the "Divisor Latch Access Bit", bit 7 of BASE+3.
BASE+0 Receiver buffer register when DLAB=0 and the operation is a read.
BASE+0 Transmitter holding register when DLAB=0 and the operation is write.
BASE+0 Divisor latch bits 0 - 7 when DLAB=1
BASE+1 Divisor latch bits 8-15 when DLAB=1.

Bytes BASE+0 and BASE+1 together form a 16-bit number, the divisor, which determines the baud rate. Set the divisor as follows:

Baud rate	Divisor	Baud rate	Divisor
50	2304	2400	48
75	1536	3600	32
110	1047	4800	24
133.5	857	7200	16
150	768	9600	12
300	384	19200	6
600	192	38400	3
1200	96	56000	2
1800	64	115200	1
2000	58	x	x

BASE+1 Interrupt Status Register (ISR) when DLAB=0
bit 0: Enable received-data-available interrupt
bit 1: Enable transmitter-holding-register-empty interrupt
bit 2: Enable receiver-line-status interrupt
bit 3: Enable modem-status interrupt

BASE+2 FIFO Control Register (FCR)
bit 0: Enable transmit and receive FIFOs
bit 1: Clear contents of receive FIFO
bit 2: Clear contents of transmit FIFO
bits 6-7: Set trigger level for receiver FIFO interrupt

Bit 7	Bit 6	FIFO trigger level
0	0	01
0	1	04
1	0	08
1	1	14

BASE+3 Line Control Register (LCR)
bit 0: Word length select bit 0
bit 1: Word length select bit 1

Bit 1	Bit 0	Word length (bits)
0	0	5
0	1	6
1	0	7
1	1	8

BASE+4 Modem Control Register (MCR)
bit 0: DTR
bit 1: RTS
BASE+5 Line Status Register (LSR)
bit 0: Receiver data ready
bit 1: Overrun error
bit 2: Parity error
bit 3: Framing error
bit 4: Break interrupt
bit 5: Transmitter holding register empty
bit 6: Transmitter shift register empty
bit 7: At least one parity error, framing error or break indication in the FIFO
BASE+6 Modem Status Register (MSR)
bit 0: Delta CTS
bit 1: Delta DSR
bit 2: Trailing edge ring indicator
bit 3: Delta received line signal detect
bit 4: CTS
bit 5: DSR
bit 6: RI
bit 7: Received line signal detect
BASE+7 Temporary data register

Appendix B

RS-485 Network

EIA RS-485 is the industry's most widely used bidirectional, balanced transmission line standard. It is specifically developed for industrial multi-drop systems that should be able to transmit and receive data at high rates or over long distances.

The specifications of the EIA RS-485 protocol are as follows:

⌘ Maximum line length per segment: 1200 meters (4000 feet)

⌘ Throughput of 10M baud and beyond -Differential transmission
(balanced lines) with high resistance against noise

⌘ Maximum 32 nodes per segment

⌘ Bi-directional master-slave communication over a single set of twisted-pair cables

⌘ Parallel connected nodes, true multi-drop

ADAM-4500 Series Controller is fully isolated and use just a single set of twisted pair wires to send and receive! Since the nodes are connected in parallel they can be freely disconnected from the host
without affecting the functioning of the remaining nodes. An industry
standard, shielded twisted pair is preferable due to the high noise ratio of the environment. When nodes communicate through the network, no sending conflicts can occur since a simple command/response sequence is used. There is always one initiator (with no address) and many slaves (with addresses). In this case, the master is a personal computer that is connected with its serial, RS-232, port to an ADAM RS-232/RS-485 converter. The slaves are the ADAM-4500 Series Controller. When systems are not transmitting data, they are in listen mode. The host computer initiates a command/response sequence with one of the systems. Commands normally contain the address of the module the host wants to communicate with. The system with the matching address carries out the command and sends its response to the host.

B.1 Basic Network Layout

Multi-drop RS-485 implies that there are two main wires in a segment. The connected systems tap from these two lines with so called drop cables. Thus all connections are parallel and connecting or disconnecting of a node doesn't affect the network as a whole. Since ADAM-4500 Series Controller use the RS-485 standard, they can connect and communicate with the host PC. The basic layouts that can be used for an RS-485 network are:

Daisychain

The last module of a segment is a repeater. It is directly connected to the main-wires thereby ending the first segment and starting the next segment. Up to 32 addressable systems can be daisychained. This limitation is a physical one. When using more systems per segment the IC driver current rapidly decreases, causing communication errors.

In total, the network can hold up to 64 addressable systems. The limitation on this number is the two-character hexadecimal address code that can address 64 combinations. The ADAM converter, ADAM repeaters and the host computer are non addressable units and therefore are not included in these numbers.

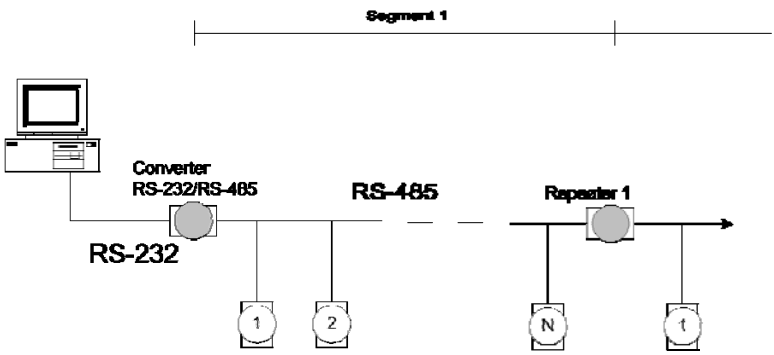


Figure B-1: *Daisychaining*

Star Layout

In this scheme the repeaters are connected to drop-down cables from the main wires of the first segment. A tree structure is the result. This scheme is not recommended when using long lines since it will cause a serious amount of signal distortion due to signal reflections in several line-endings.

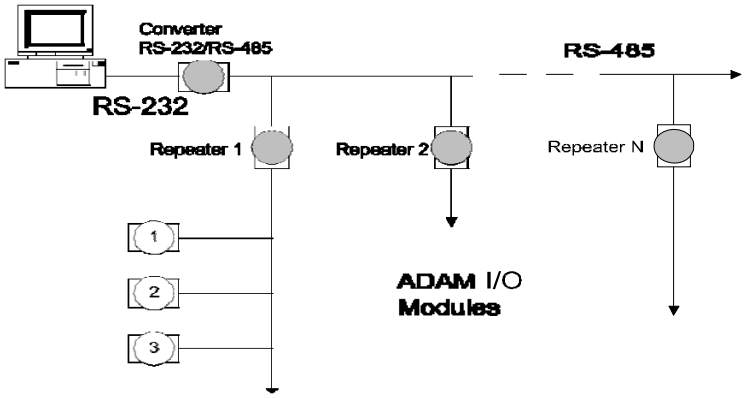


Figure B-2: *Star structure*

Random

This is a combination of daisychain and hierarchical structure.

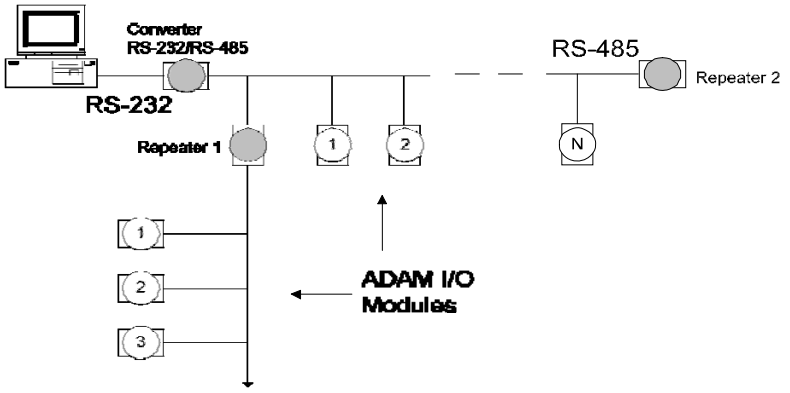


Figure B-3: *Random structure*

B.2 Line Termination

Each discontinuity in impedance causes reflections and distortion. When an impedance discontinuity occurs in the transmission line the immediate effect is signal reflection. This will lead to signal distortion. Especially at line ends this mismatch causes problems. To eliminate this discontinuity, terminate the line with a resistor.

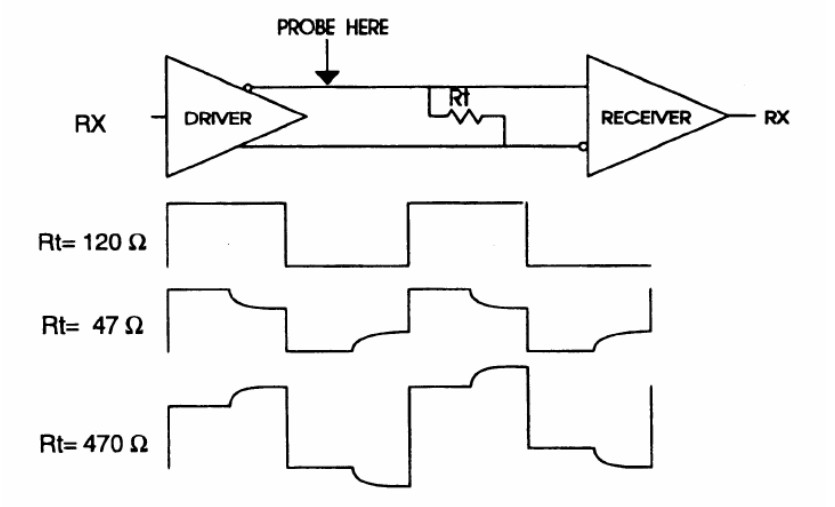


Figure B-4: *Signal distortion*

The value of the resistor should be a close as possible to the characteristic impedance of the line. Although receiver devices add some resistance to the whole of the transmission line, normally it is sufficient to the resistor impedance should equal the characteristic impedance of the line.

Example: Each input of the receivers has a nominal input impedance of 18 k feeding into a diode transistor- resistor biasing network that is equivalent to an 18 input resistor tied to a common k mode voltage of 2.4 V. It is this configuration, which provides the large common range of the receiver required for RS-485 systems!

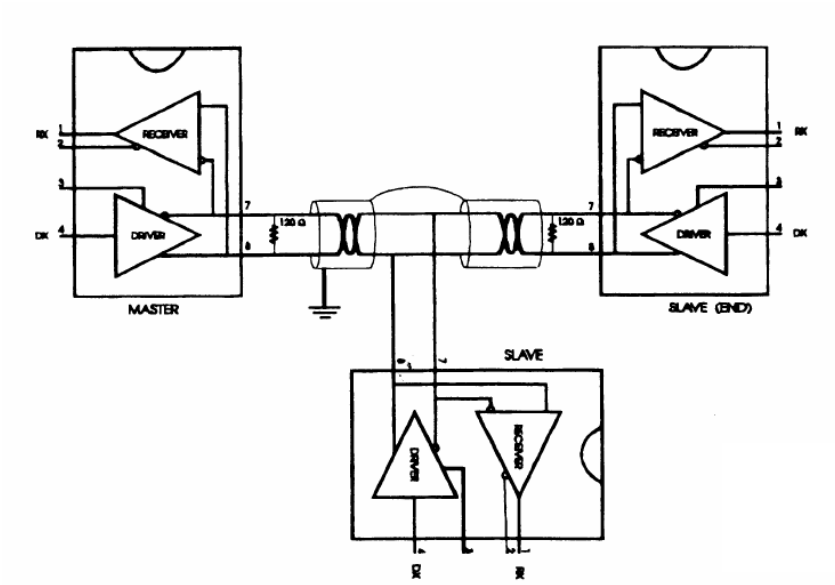


Figure B-5: *Termination resistor locations*

Because each input is biased to 2.4 V, the nominal common mode voltage of balanced RS-485 systems, the 18 k on the input can be taken as being in series across the input of each individual receiver. If thirty of these receivers are put closely together at the end of the transmission line, they will tend to react as thirty 36k resistors in parallel with the termination resistor. The overall effective resistance will need to be close to the characteristics of the line. The effective parallel receiver resistance R_P will therefore be equal to:

$R_P = 36 \times 10^3/30 = 1200\Omega$

While the termination receptor R_T will equal:
$R_T = R_O / [1 - R_O/R_P]$

Thus for a line with a characteristic impedance of 100 resistor R_T =
$100/[1 - 100/1200] = 110\Omega$

Since this value lies within 10% of the line characteristic impedance.

Thus as already stated above the line termination resistor R_T will normally equal the characteristic impedance Z_o. The star connection causes a multitude of these discontinuities since there are several transmission lines and is therefore not recommend.

Note: The recommend method wiring method, that causes a minimum amount of reflection, is daisy chaining where all receivers tapped from one transmission line needs only to be terminated twice.

B.3 RS-485 Data Flow Control

The RS-485 standard uses a single pair of wires to send and receive data. This line sharing requires some method to control the direction of the data flow. RTS (Request To Send) and CTS (Clear To Send) are the most commonly used methods.

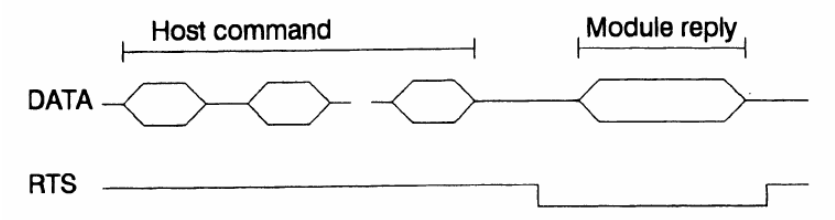


Figure B-6: *RS-485 data flow control with RTS*

Intelligent RS-485 Control

ADAM-4510 and ADAM-4520 are both equipped with an I/O circuit which can automatically sense the direction of the data flow. No handshaking with the host (like RTS, Request to Send) is necessary to receive data and forward it in the correct direction. You can use any software written for half-duplex RS-232 with an ADAM network without modification. The RS-485 control is completely transparent to the user.

Appendix C
Grounding Reference

Field Grounding and Shielding Application

Overview

Unfortunately, it's impossible to finish a system integration task at one time. We always meet some trouble in the field. A communication network or system isn't stable, induced noise or equipment is damaged or there are storms. However, the most usual issue is just simply improper wiring, ie, grounding and shielding. You know the 80/20 rule in our life: we spend 20% time for 80% work, but 80% time for the last 20% of the work. So is it with system integration: we pay 20% for Wire / Cable and 0% for Equipment. However, 80% of reliability depends on Grounding and Shielding. In other words, we need to invest more in that 20% and work on these two issues to make a highly reliable system. This application note brings you some concepts about field grounding and shielding. These topics will be illustrated in the following pages.

1. Grounding

1.1 The 'Earth' for reference

1.2 The 'Frame Ground' and 'Grounding Bar'

1.3 Normal Mode and Common Mode

1.4 Wire impedance

1.5 Single Point Grounding

2. Shielding

2.1 Cable Shield

2.2 System Shielding

3. Noise Reduction Techniques

4. Check Point List

C.1 Grounding

C-1.1 The 'Earth' for reference

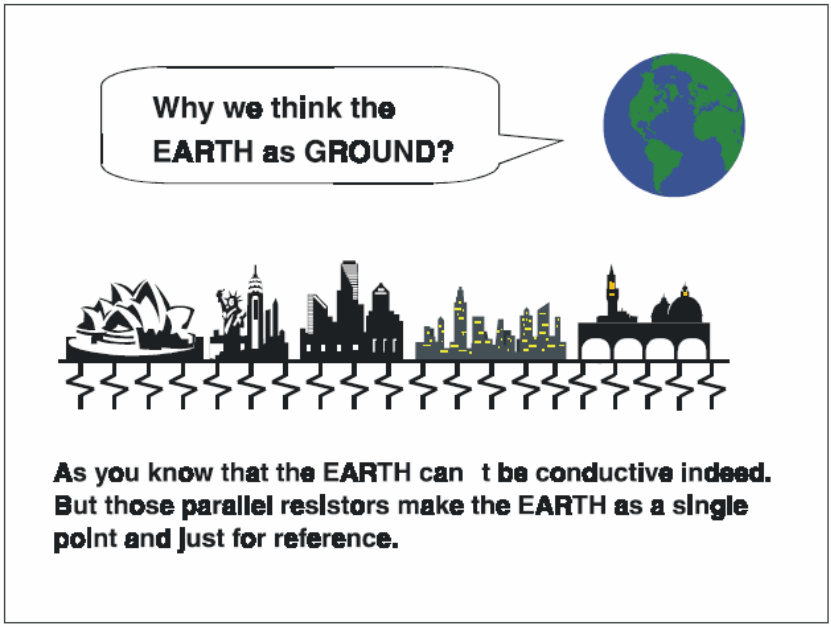

Why we think the
EARTH as GROUND?
As you know that the EARTH can t be conductive indeed.
But those parallel resistors make the EARTH as a single
point and just for reference.

Figure C-1: Think the EARTH as GROUND.

As you know, the EARTH cannot be conductive. However, all buildings lie on, or in, the EARTH. Steel, concrete and associated cables (such as lighting arresters) and power system were connected to EARTH. Think of them as resistors. All of those infinite parallel resistors make the EARTH as a single reference point.

C-1.2 The 'Frame Ground' and 'Grounding Bar'

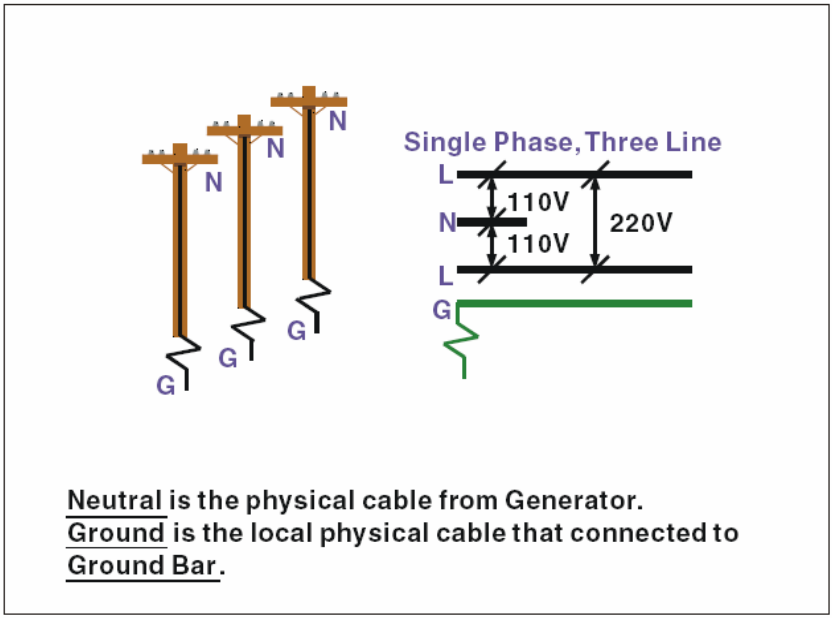


Figure C-2: Grounding Bar.

Grounding is one of the most important issues for our system. Just like Frame Ground of the computer, this signal offers a reference point of the electronic circuit inside the computer. If we want to communicate with this computer, both Signal Ground and Frame Ground should be connected to make a reference point of each other's electronic circuit. Generally speaking, it is necessary to install an individual grounding bar for each system, such as computer networks, power systems, telecommunication networks, etc. Those individual grounding bars not only provide the individual reference point, but also make the earth our ground!

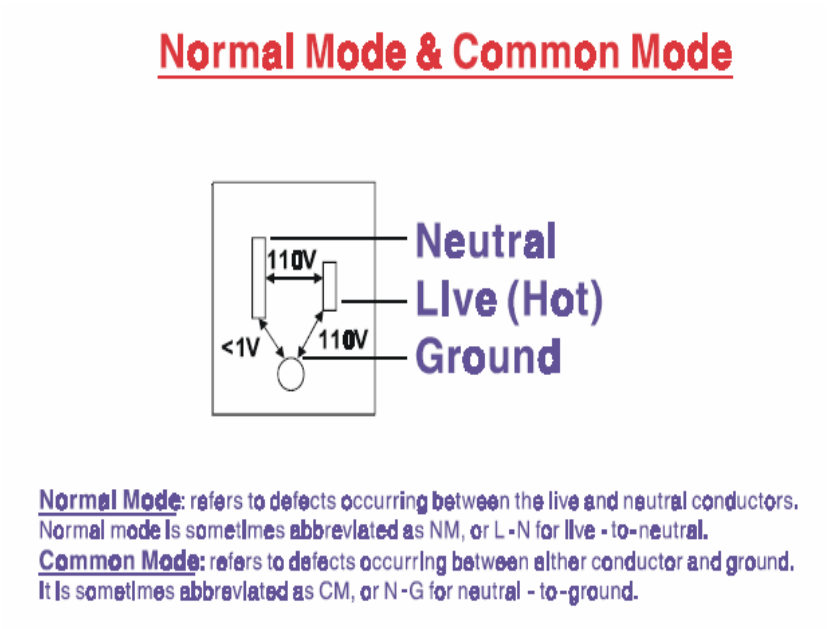

Normal Mode & Common Mode
110V
<1V
110V
Neutral
LIve (Hot)
Ground
Normal Mode: refers to defects occurring between the live and neutral conductors.
Normal mode Is sometImes abbrevlated as NM, or L-N for live-to-neutral.
Common Mode: refers to defects occurrIng between alther conductor and ground.
It Is sometImes abbrevlated as CM, or N-G for neutral-to-ground.

Figure C-3: Normal mode and Common mode.

C-1.3 Normal Mode and Common Mode

Have you ever tried to measure the voltage between a live circuit and a concrete floor? How about the voltage between neutral and a concrete floor? You will get nonsense values. 'Hot' and 'Neutral' are just relational signals: you will get 110VAC or 220VAC by measuring these signals. Normal mode and common mode just show you that the Frame Ground is the most important reference signal for all the systems and equipments.

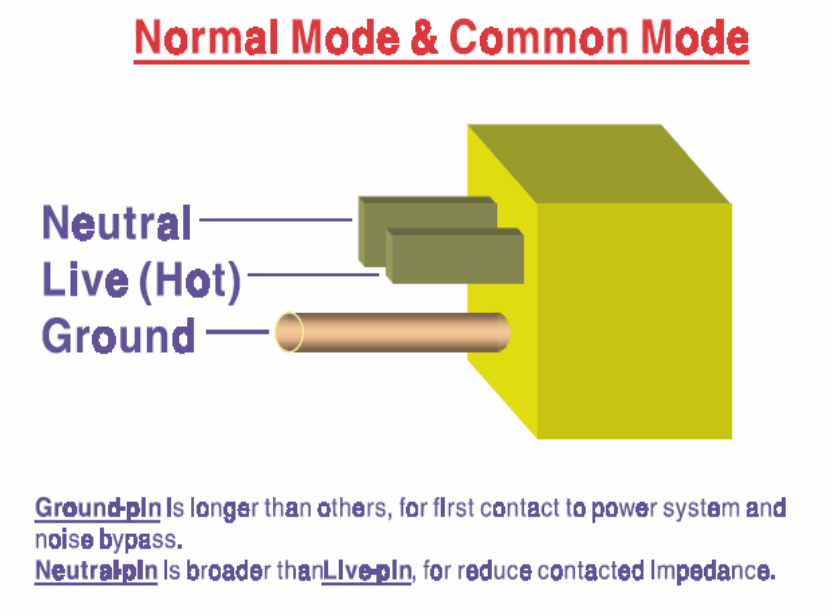

Normal Mode & Common Mode
Neutral
Live (Hot)
Ground
Ground-pin is longer than others, for first contact to power system and noise bypass.
Neutral-pin is broader than Live-pin, for reduce contacted impedance.

Figure C-4: Normal mode and Common mode.

- Ground-pin is longer than others, for first contact to power system and noise bypass.
- Neutral-pin is broader than Live-pin, for reducing contact impedance.

C-1.4 Wire impedance

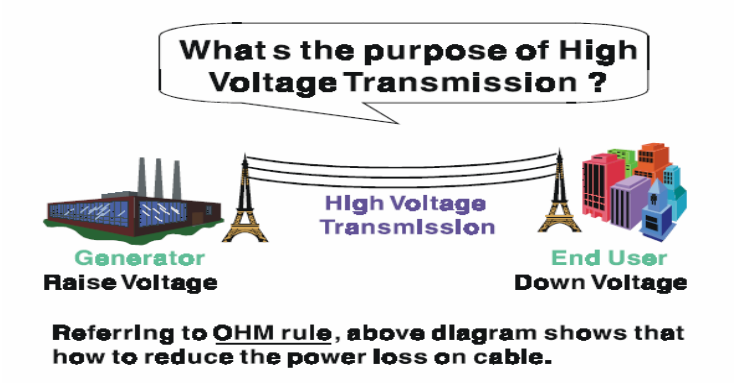

What s the purpose of High Voltage Transmission ?
High Voltage Transmission
Generator
Raise Voltage
End User
Down Voltage
Referring to OHM rule, above diagram shows that how to reduce the power loss on cable.

Figure C-5: The purpose of high voltage transmission

• What's the purpose of high voltage transmission? We have all seen high voltage transmission towers. The power plant raises the voltage while generating the power, then a local power station steps down the voltage. What is the purpose of high voltage transmission wires? According to the energy formula, P = V * I, the current is reduced when the voltage is raised. As you know, each cable has impedance because of the metal it is made of. Referring to Ohm's Law, (V = I * R) this decreased current means lower power losses in the wire. So, high voltage lines are for reducing the cost of moving electrical power from one place to another.

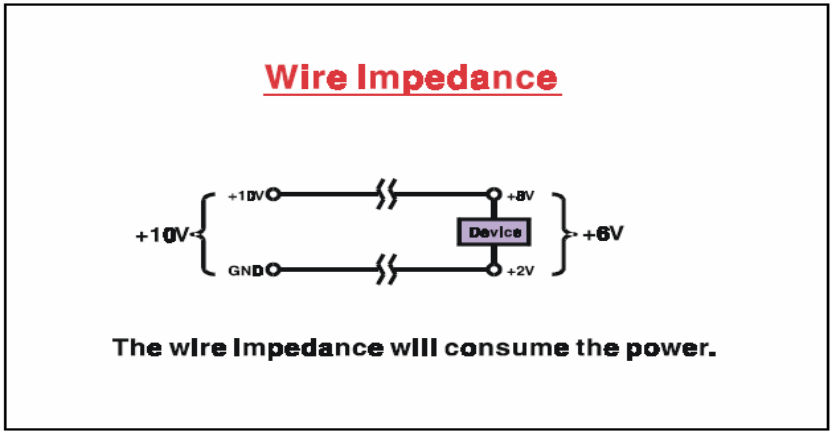


Figure C-6: wire impedance.

C-1.5 Single Point Grounding

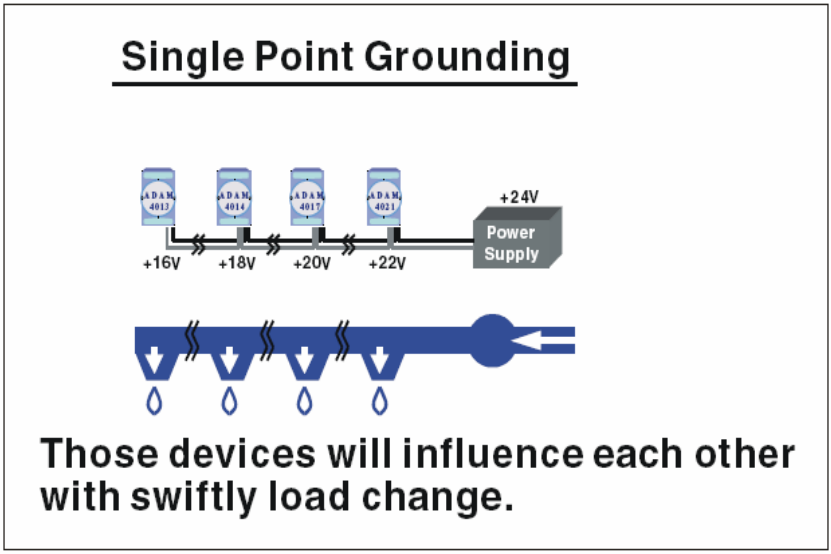


Figure C-7: Single point grounding. (1)

• What's Single Point Grounding? Maybe you have had an unpleasant experience while taking a hot shower in winter. Someone turns on a hot water faucet somewhere else. You will be impressed with the cold water! The bottom diagram above shows an example
of how devices will influence each other with swift load change. For example, normally we turn on all the four hydrants for testing. When you close the hydrant 3 and hydrant 4, the other two hydrants will get more flow. In other words, the hydrant cannot keep a constant flow rate.

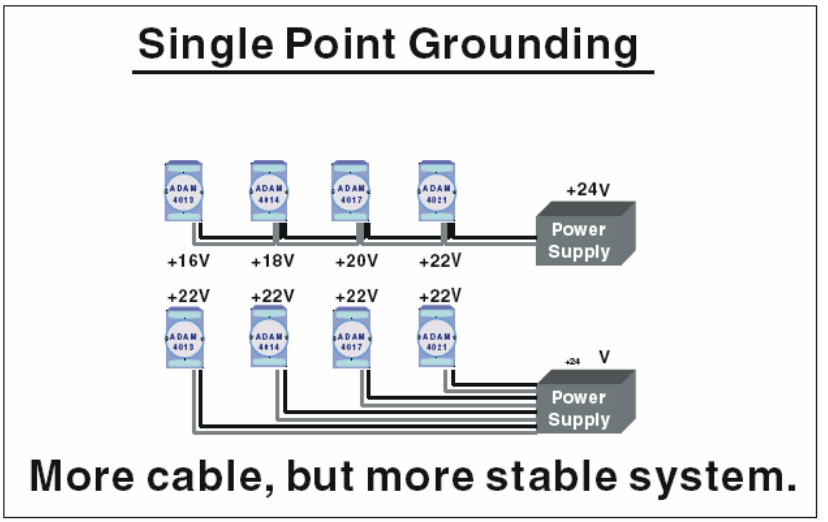


Figure C-8: Single point grounding. (2)

The above diagram shows you that a single point grounding system will be a more stable system. If you use thin cable for powering these devices, the end device will actually get lower power. The thin cable will consume the energy.

C.2 Shielding

C-2.1 Cable Shield

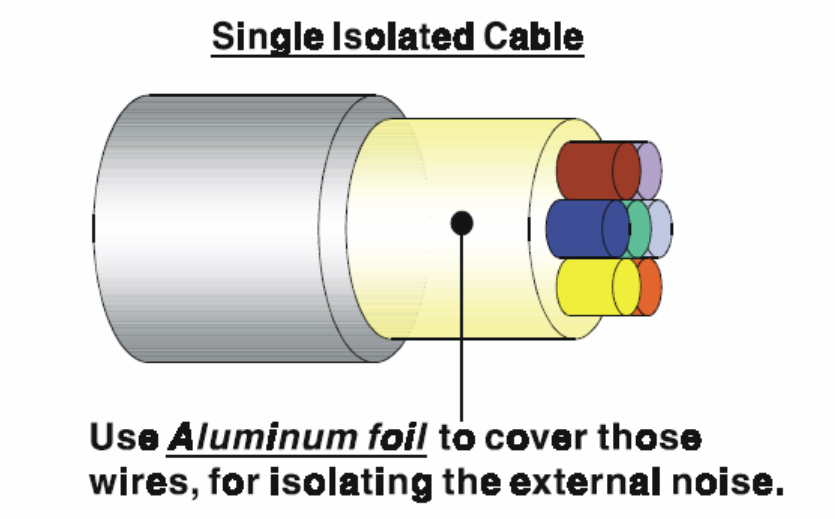


Figure C-9: Single isolated cable

• Single isolated cable
The diagram shows the structure of an isolated cable. You see the isolated layer which is spiraled Aluminum foil to cover the wires. This spiraled structure makes a layer for shielding the cables from external noise.

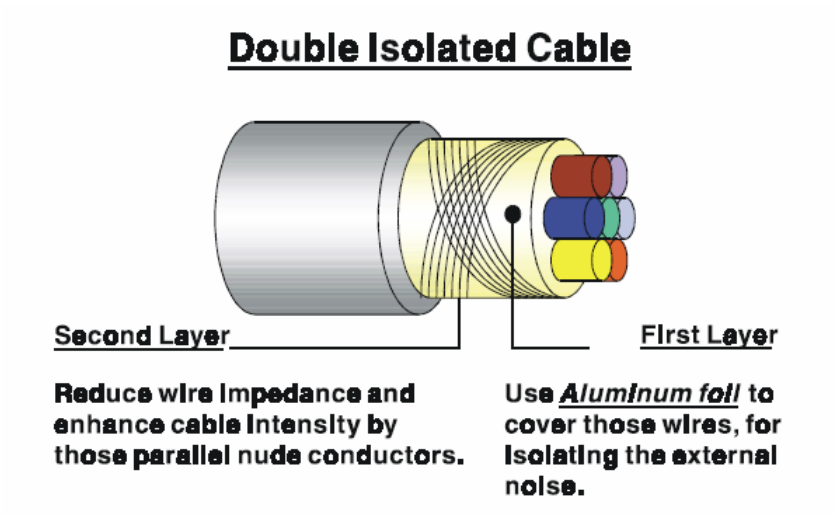

Double Isolated Cable
Second Layer
First Layer
Reduce wire impedance and enhance cable intensity by those parallel nude conductors.
Use Aluminum foil to cover those wires, for isolating the external noise.

Figure C-10: Double isolated cable

- Double isolated cable

Figure 10 is an example of a double isolated cable. The first isolating layer of spiraled aluminum foil covers the conductors. The second isolation layer is several bare conductors that spiral and cross over the first shield layer. This spiraled structure makes an isolated layer for reducing external noise. Additionally, follow these tips just for your reference.

- The shield of a cable cannot be used for signal ground. The shield is designed for carrying noise, so the environment noise will couple and interfere with your system when you use the shield as signal ground.
- The higher the density of the shield - the better, especially for communication network.
- Use double isolated cable for communication network / AI / AO.
- Both sides of shields should be connected to their frame while inside the device. (for EMI consideration)
- Don't strip off too long of plastic cover for soldering.

C-2.2 System Shielding

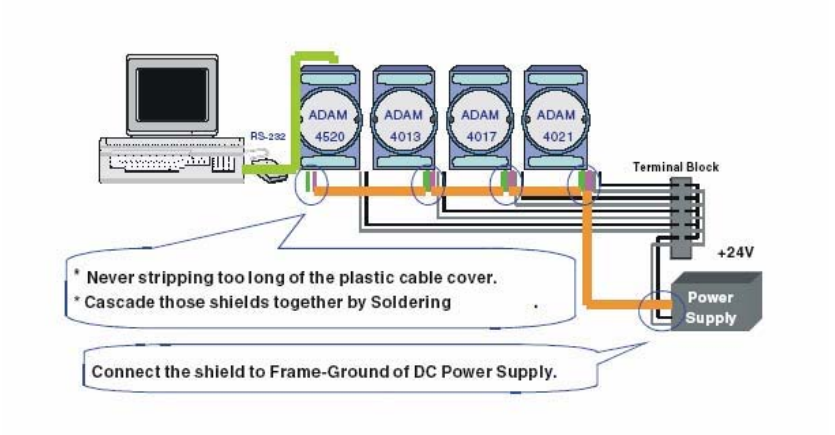


Figure C-11: System Shielding

• Never stripping too much of the plastic cable cover. This is improper and can destroy the characteristics of the Shielded-Twisted-Pair cable. Besides, the bare wire shield easily conducts the noise.

• Cascade these shields together by soldering. Please refer to following page for further detailed explanation.

• Connect the shield to Frame Ground of DC power supply to force the conducted noise to flow to the frame ground of the DC power supply. (The 'frame ground' of the DC power supply should be connected to the system ground)

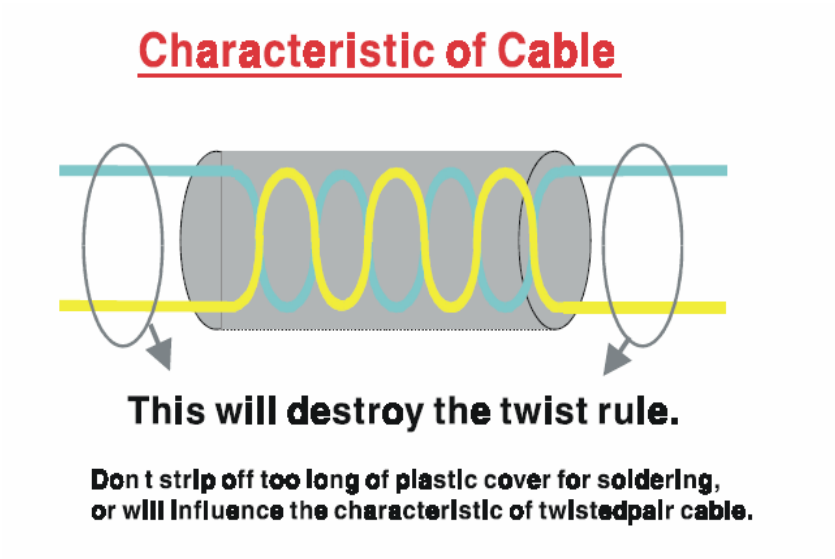

Characteristic of Cable
This will destroy the twist rule.
Don t strip off too long of plastic cover for soldering,
or will influence the characteristic of twistedpair cable.

Figure C-12: The characteristic of the cable

• The characteristic of the cable
Don't strip off too much insulation for soldering. This could change the effectiveness of the Shielded-Twisted-Pair cable and open a path to introduce unwanted noise.

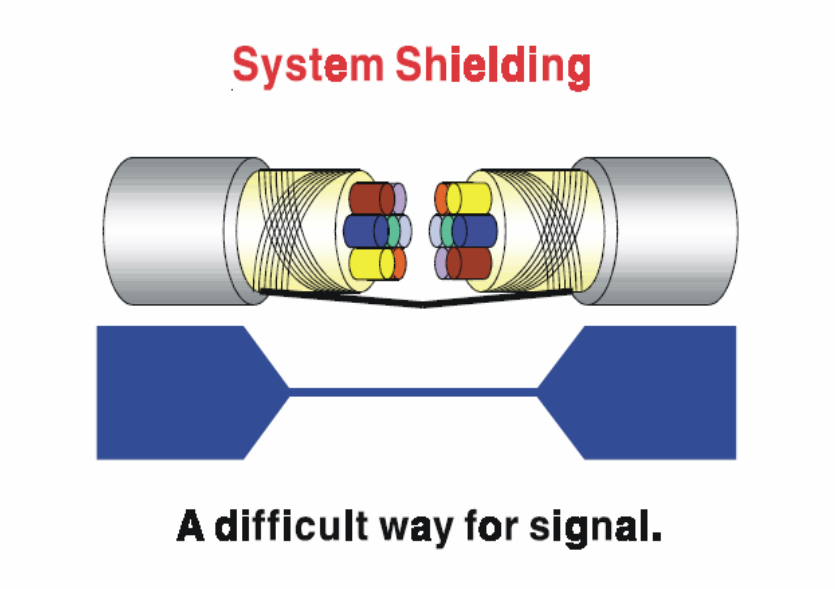


Figure C-13: System Shielding (1)

• Shield connection (1)
If you break into a cable, you might get in a hurry to achieve your goal. As in all electronic circuits, a signal will use the path of least resis- tance. If we make a poor connection between these two cables we will make a poor path for the signal. The noise will try to find another path for easier flow.

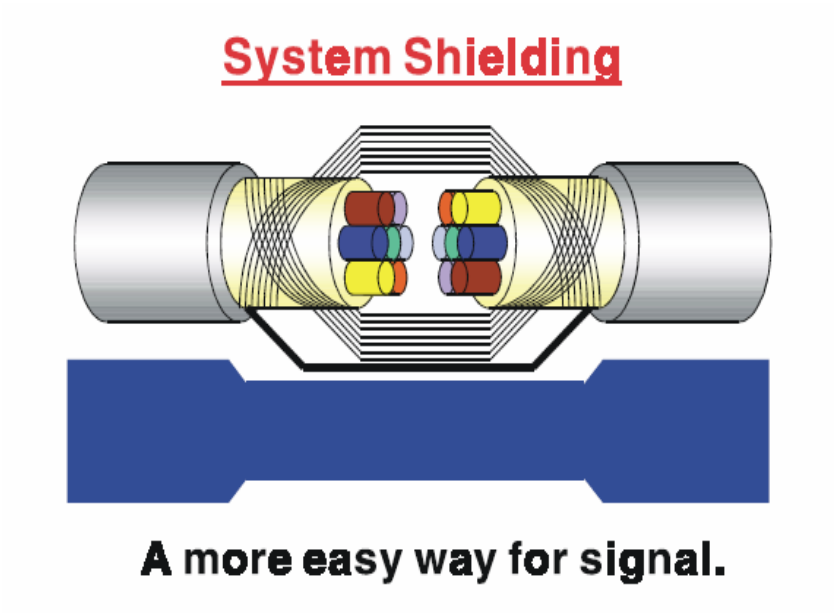


Figure C-14: System Shielding (2)

• Shield connection (2)

The previous diagram shows you that the fill soldering just makes an easier way for the signal.

C.3 Noise Reduction Techniques

- Isolate noise sources in shielded enclosures.
- Place sensitive equipment in shielded enclosure and away from computer equipment.
- Use separate grounds between noise sources and signals.
- Keep ground/signal leads as short as possible.
- Use Twisted and Shielded signal leads.
- Ground shields on one end ONLY while the reference grounds are not the same.
- Check for stability in communication lines.
- Add another Grounding Bar if necessary.
- The diameter of power cable must be over 2.0 mm_2.
- Independent grounding is needed for A/I, A/O, and communication network while using a jumper box.
- Use noise reduction filters if necessary. (TVS, etc)
- You can also refer to FIPS 94 Standard. FIPS 94 recommends that the computer system should be placed closer to its power source to eliminate load-induced common mode noise.

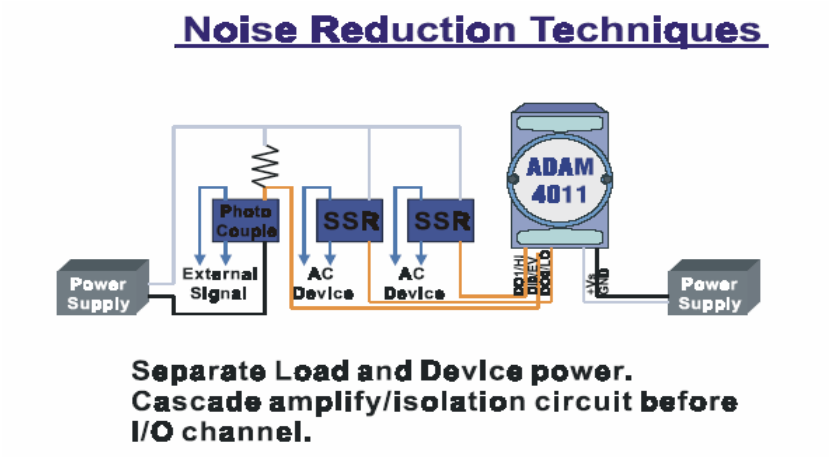


Figure C-15: Noise Reduction Techniques

C.4 Check Point List

- Follow the single point grounding rule?
- Normal mode and common mode voltage?
- Separate the DC and AC ground?
- Reject the noise factor?
- The shield is connected correctly?
- Wire size is correct?
- Soldered connections are good?
- The terminal screw are tight?